KNX CONTROL

BENUTZERHANDBUCH SCRIPTING - MODUL

VERSION 1.1

ALLGEMEINE INFORMATIONEN

DIVUS GmbH
Pillhof 51
I-39057 Eppan (BZ)

Betriebsanleitungen, Handbücher und Software sind urheberrechtlich geschützt. Alle Rechte bleiben vorbehalten. Das Kopieren, Vervielfältigen, Übersetzen, Umsetzen im Ganzen oder in Teilen ist nicht gestattet. Eine Ausnahme gilt für die Anfertigung einer Sicherungskopie der Software für den eigenen Gebrauch.

Änderungen des Handbuchs behalten wir uns ohne Vorankündigung vor. Die Fehlerfreiheit und Richtigkeit der in diesem Dokument und auf den mitgelieferten Speichermedien enthaltenen Daten können wir nicht garantieren. Anregungen zu Verbesserungen sowie Hinweise auf Fehler sind uns jederzeit willkommen. Die Vereinbarungen gelten auch für die speziellen Anhänge zu diesem Handbuch.

Die Bezeichnungen in diesem Dokument können Marken sein, deren Benutzung durch Dritte für eigene Zwecke die Rechte der Inhaber verletzen können.

Benutzerhinweise: Bitte lesen Sie das Handbuch **vor** dem ersten Einsatz und bewahren Sie es zur späteren Verwendung sorgfältig auf.

Zielgruppe: Das Handbuch ist für Anwender mit Vorkenntnissen in der PC- und Automatisierungstechnik geschrieben.

DARSTELLUNGSKONVENTIONEN

[TASTE]	Tasteneingaben des Benutzers werden in eckigen Klammern dargestellt, z.B. [STRG] oder [ENTF]
Courier	Bildschirmausgaben werden in der Schriftart Courier beschrieben, z.B. C:\>
Courier Fett	Tastatureingaben durch den Benutzer sind in Schriftart Courier fett beschrieben, z.B. C:\>**DIR**
„…"	Namen von auszuwählenden Schaltflächen, Menüs oder anderen Bildschirmelementen werden in *„Gänsefüßchen"* wiedergegeben.
Piktogramme	Im Handbuch sind folgende Piktogramme zur Kennzeichnung bestimmter Textabschnitte verwendet:
	Achtung! Möglicherweise gefährliche Situation. Sachschäden können die Folge sein.
	Notizen Tipps und ergänzende Hinweise

INHALTSVERZEICHNIS:

1 EINLEITUNG - 7 -

1.1 WAS IST KNXCONTROL - 7 -

1.2 WAS BEZWECKT DIESES HANDBUCH - 7 -

1.3 ANFORDERUNGEN - 8 -

2 ALLGEMEINE ÜBERSICHT - 9 -

2.1 SCRIPTS - 9 -

2.2 RUN-SCRIPTS - 9 -

2.3 BIBLIOTHEKEN - 11 -

2.4 INPUTS UND OUTPUTS - 11 -

2.5 DEBUG, INFO UND FEHLER - 13 -

2.6 EINBINDUNG VON SCRIPTS - 14 -

3 SCRIPTS - 15 -

3.1 EINLEITUNG - 15 -

3.2 VERWALTUNGSFENSTER DER SCRIPTS - 15 -

3.3 DARSTELLUNG / BEARBEITUNG EINES SCRIPTS - 17 -

3.4 BEISPIEL - 17 -

4 RUN-SCRIPTS - 18 -

4.1 EINLEITUNG - 18 -

4.2 ERSTELLUNG EINES RUN-SCRIPTS - 18 -

4.2.1 EINGÄNGE - 20 -

4.2.2 AUSGÄNGE - 21 -

4.3 DEBUG - 21 -

4.4 AUSFÜHRUNG IM HINTERGRUND - 22 -

4.5 DARSTELLUNG IN DER VISUALISIERUNG - 23 -

5 "OBJECT" - BIBLIOTHEK - 24 -

5.1 EINLEITUNG - 24 -

5.2 EINBINDUNG DER BIBLIOTHEK - 24 -

5.3 LADEN EINES OBJEKTS - 25 -

5.4 VERKNÜPFUNGEN MIT ANDEREN OBJEKTEN - 26 -

5.5 BEDIENUNG EINES OBJEKTS - 31 -

5.6 AKTUALISIERUNG VON OBJEKTEN IN DER DATENBANK - 33 -

6 "SURROUNDING" - BIBLIOTHEK - 34 -

6.1 EINLEITUNG - 34 -

6.2 EINBINDUNG DER BIBLIOTHEK - 34 -

6.3 UMGEBUNG DES RUN-SCRIPTS - 35 -

6.4 BEISPIELE - 35 -

7 "SERIAL" - BIBLIOTHEK - 37 -

7.1 EINLEITUNG - 37 -

7.2 EINBINDUNG DER BIBLIOTHEK - 37 -

7.3 INITIALISIERUNG DER SCHNITTSTELLE - 38 -

7.4 SCHREIBVORGÄNGE - 38 -

7.5 LESEVORGÄNGE - 39 -

7.6 DIREKTER ZUGRIFF AUF DIE SCHNITTSTELLE - 39 -

8 "MODBUS" - BIBLIOTHEK - 40 -

8.1 EINLEITUNG - 40 -

8.2 EINBINDUNG DER BIBLIOTHEK - 40 -

8.3 ZUWEISUNG DES "MODBUS SLAVE" - GERÄTES - 41 -

8.4 AUSLESEN DER REGISTER - 41 -

8.5 AUSLESEN DER COILS - 41 -

8.6 SCHREIBEN VON REGISTERN - 42 -

8.7 SCHREIBEN VON COILS - 42 -

8.8 LESEN UND SCHREIBEN VON REGISTERN - 43 -

8.9 UMWANDLUNG DER EINGELESENEN WERTE - 43 -

8.9.1 4 BYTE - WERTE - 43 -

8.9.2 2 BYTE - WERTE - 44 -

9 "SONOS" – BIBLIOTHEK - 45 -

9.1 EINLEITUNG - 45 -

9.2 EINBINDUNG DER BIBLIOTHEK - 45 -

9.3 ZUWEISUNG DES "SONOS"-GERÄTS - 46 -

9.4 BEDIENUNG DES "SONOS"-GERÄTS - 46 -

9.5 LAUTSTÄRKE-REGELUNG - 47 -

9.6 WIEDERGABE-MODUS - 48 -
9.7 MULTIMEDIA-INFOS - 48 -
9.8 KONFIGURATIONSBEISPIEL - 48 -
9.8.1 ERSTELLUNG KOMPLEXES OBJEKT - 49 -
9.8.2 ERSTELLUNG UND VERKNÜPFUNG DER RUN-SCRIPTS - 50 -
10 "DUNE" – BIBLIOTHEK - 52 -
10.1 EINLEITUNG - 52 -
10.2 EINBINDUNG DER BIBLIOTHEK - 52 -
10.3 ZUWEISUNG DES "DUNE"-GERÄTS - 53 -
10.4 ALLGEMEINE BEFEHLE ZUM "DUNE"-GERÄT - 53 -
10.5 BEDIENUNG DES "DUNE"-GERÄTS - 54 -
10.6 EMULATION FERNBEDIENUNG - 54 -
10.7 WIEDERGABE-STEUERUNG - 55 -
11 "MESSAGES" – BIBLIOTHEK - 56 -
11.1 EINLEITUNG - 56 -
11.2 EINBINDUNG DER BIBLIOTHEK - 56 -
11.3 VERSENDUNG EINER ONSCREEN-NACHRICHT - 57 -
11.4 VERSENDUNG EINER E-MAIL-NACHRICHT - 57 -
12 BEISPIELE - 58 -
12.1 EINLEITUNG - 58 -
12.2 LOGIC.AND - 58 -
12.3 LOGIC.OR - 59 -
12.4 SERIAL.WRITER_GENERIC - 59 -
12.5 DEMUX.STATUSBYTE - 60 -
12.6 MATH.SUM - 62 -
12.7 MATH.PRODUCT - 62 -
12.8 MODBUS.READCOILS - 63 -
12.9 MODBUS.READREGISTERS - 66 -
12.10 MODBUS.WRITECOILS - 67 -
12.11 SONOS.GENERIC - 68 -
12.12 SONOS.SETVOLUME - 69 -
12.13 SONOS.GETVOLUME - 69 -

12.14 SONOS.PLAYPAUSE - 69 -
12.15 SONOS.PREVNEXT - 70 -
12.16 SONOS.SETPLAYMODE - 70 -
12.17 SONOS.GETINFO - 70 -
12.18 DUNE.ONOFF - 71 -
12.19 DUNE.PLAYPAUSE - 71 -
12.20 DUNE.SETVOLUME - 72 -
12.21 DUNE.IRCOMMAND - 72 -
12.22 DEWPOINT - 72 -
13 ANHANG - 73 -
13.1 FUNKTIONS-OBJEKTTYPEN - 73 -
13.2 WEB-OBJEKTTYPEN - 75 -
I. RELEASE NOTES - 76 -
II. NOTIZEN - 77 -

1 EINLEITUNG

1.1 WAS IST KNXCONTROL

KNXCONTROL ist eine Produktfamilie zur Überwachung und Visualisierung von Home & Building Automation Anlagen, welche auf Basis des weltweiten KNX-Standards realisiert worden sind. Die **KNX**CONTROL-Produkte erlauben die Verwaltung sämtlicher Funktionen der Anlage durch Browser-Zugriff über beliebige PCs / MACs / Touchpanels, Tablets oder Smartphones der letzten Generation, sowohl innerhalb des Netzwerks als auch remote via Internet. Über die Homepage www.divus.eu können sämtliche Datenblätter, Broschüren und technische Handbücher, welche zur Inbetriebnahme und Verwendung der **KNX**CONTROL-Produkte benötigt werden, kostenlos bezogen werden.

1.2 WAS BEZWECKT DIESES HANDBUCH

Dieses Handbuch erläutert die Funktionalität des Scripting-Moduls der **KNX**CONTROL-Geräte, über welches deren Basis-Funktionen nach Belieben erweitert werden können. Folgende Liste zeigt einige Verwendungsbeispiele:

- Interaktion mit den Objekten der Software (Status auslesen, Werte evaluieren, Befehle versenden)
- Versendung personalisierter Benachrichtigungen
- Interaktion mit dem Betriebssystem
- Versendung / Empfang von Befehlen über LAN und serielle Schnittstelle

Das Handbuch führt Sie Schritt für Schritt durch die Erstellung eines personalisierten Scripts sowie dessen Ausführung innerhalb der Visualisierung des **KNX**CONTROL-Produkts. Zudem wird im Detail auf die verschiedenen, vorinstallierten Beispiel-Scripts eingegangen, welche bereits einen guten Einblick in die Möglichkeiten des Scripting-Moduls geben.

1.3 ANFORDERUNGEN

Um die Scripting-Funktionen der **KNX**CONTROL-Produkte optimal einsetzen zu können, benötigen Sie:

- ein **KNX**CONTROL-Gerät inkl. aktivem Scripting-Modul
- gute Kenntnisse im Umgang mit **KNX**CONTROL / OPTIMA
- gute Kenntnisse der ETS sowie der KNX-Technologie allgemein
- Basis-Kenntnisse der Programmiersprache PHP

2 ALLGEMEINE ÜBERSICHT

2.1 SCRIPTS

Die SCRIPTS sind mit PHP-Code gefüllte Programm-Dateien, welche direkt im Flash-Speicher des **KNX**CONTROL-Gerätes abgelegt werden und über einen entsprechenden Bereich in der Konfigurationsebene (VERWALTUNG) der OPTIMA-Oberfläche bearbeitet werden können. Diese Dateien verfügen über ein eigenes Eigenschaftsfenster zur Erstellung / Bearbeitung, über welches auch ein Export der Scripts zur Weiterverwendung auf anderen Geräten ermöglicht wird.

Diese Dateien bleiben auch beim Rücksetzen der Datenbank des **KNX**CONTROL-Gerätes bestehen. Da sie jedoch nicht Teil des Datenbank-Backups sind, müssen Sie bei Bedarf manuell von einem Gerät zum nächsten übertragen werden.

Die SCRIPTS müssen nach den Richtlinien und Anleitungen in diesem Handbuch erstellt werden, unter Verwendung der zur Verfügung gestellten Bibliotheken. Jeglicher PHP-Code, welcher nicht den festgelegten Bestimmungen entspricht, kann die korrekte Funktionalität des **KNX**CONTROL-Gerätes gefährden.

Ihr **KNX**CONTROL-Gerät beinhaltet bereits mehrere Beispiel-Scripts, welche eine Übersicht der zur Verfügung stehenden Funktionen geben sollen. Die direkte Bearbeitung dieser Beispiel-Scripts ist nicht möglich, wohl aber das Kopieren und das anschließende Bearbeiten der Kopie.

2.2 RUN-SCRIPTS

Nach Erstellung eines Scripts (entweder durch Kopieren eines Beispiel-Scripts oder durch Neuerstellung) muss dieses ein oder mehreren "Run-Scripts" zugewiesen werden; diese besonderen Objekte haben die Aufgabe, den in der Script-Datei enthaltenen Code jedes Mal auszuführen,

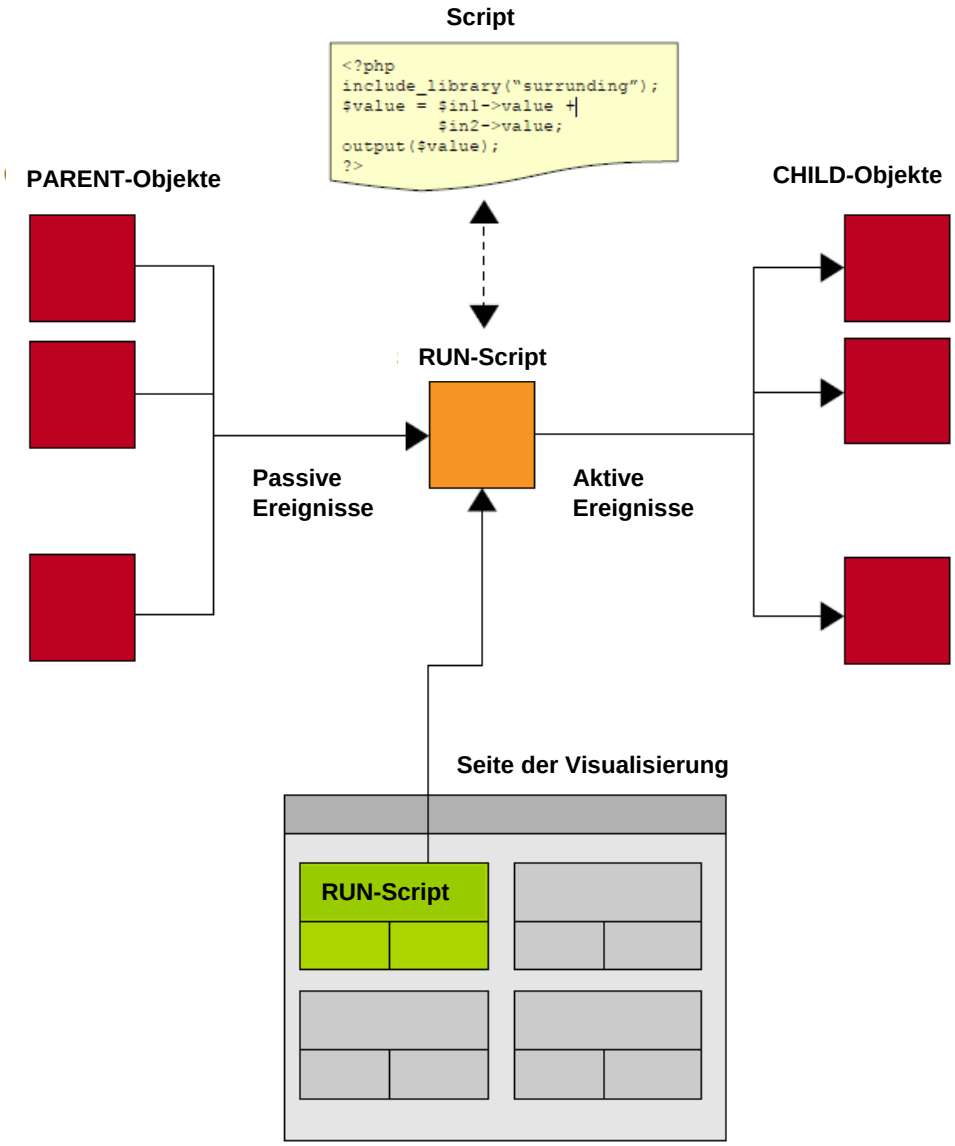
- wenn das Run-Script manuell über die entsprechende "Box" in den Seiten der Visualisierung ausgeführt wird.
- wenn der Wert eines Objektes verändert wurde, welches als EINGANG (PASSIVES EREIGNIS) des Run-Scripts definiert worden ist, wie auch für alle anderen Objekttypen in der OPTIMA-Oberfläche gültig.

Die SCRIPTS können nicht direkt ausgeführt werden, sondern immer nur über ein verknüpftes RUN-SCRIPT, welches sich wie jedes andere Objekt der Visualisierung charakterisieren lässt: es besitzt ein Symbol und einen Wert, kann in Räumlichkeiten oder Funktions-Seiten dargestellt werden, kann mit passiven und aktiven Ereignissen verknüpft werden, einem Szenario zugewiesen werden usw.

Die Vorteile dieser in SCRIPTS und RUN-SCRIPTS aufgeteilten Struktur sind:

- die Möglichkeit, die SCRIPTS gleich mehreren RUN-SCRIPTS zuzuweisen und somit verschiedene Objekte auf dieselbe Art und Weise auszuwerten / zu kontrollieren / zu bedienen.
- Die Möglichkeit, die SCRIPTS unabhängig von der Verwendung im aktuellen Projekt zu exportieren und somit auch auf Geräten mit anderen Projekten einzusetzen

Folgendes Schema soll den Informationsaustausch zwischen RUN-SCRIPT und den verknüpften Ereignissen / Objekten sowie den Datenfluss bei Bedienung grafisch darstellen:



2.3 BIBLIOTHEKEN

Jedes **KNX**CONTROL-Gerät verfügt über eine Reihe von Programm-Bibliotheken, welche bereits diverse Funktionen zur Interaktion mit den verknüpften Objekten zur Verfügung stellen. Die Bibliotheken sind nicht automatisch mit den Scripts verbunden, sondern müssen über folgenden Befehl explizit eingebunden werden:

```
include_library(<Name der Bibliothek>);
```

Beispiel:

```
include_library("serial");
```

Da das Einbinden von Bibliotheken einen Arbeits- und somit Zeitaufwand bei der Ausführung des Scripts bedeuten, wird empfohlen, nur jene Bibliotheken einzubinden, die auch effektiv vom Script benötigt werden. Die Bibliotheken können ihrerseits von weiteren Bibliotheken abhängen; in diesem Fall müssen die weiteren Bibliotheken nicht explizit eingebunden werden, da sie von der Hauptbibliothek automatisch mit eingebunden werden.

Folgende Tabelle zeigt die verfügbaren Bibliotheken sowie deren Abhängigkeiten auf. Die einzelnen Bibliotheken werden im Verlauf des Handbuchs noch genauer erklärt:

Bibliothek	Beschreibung	Abhängigkeit
db	Erlaubt die Interaktion mit der Datenbank des **KNX**CONTROL-Gerätes (Ausführung von SQL-Befehlen zum Abrufen / Bearbeiten / Einfügen / Löschen von Werten)	
object	Erlaubt die Interaktion mit den versch. Objekten der Visualisierung (Auslesen von aktuellen Werten, Veränderung der Eigenschaften, Versendung von Befehlen)	db
surrounding	Erlaubt die direkte Interaktion mit dem verknüpften Run-Script sowie den passiven und aktiven Ereignissen	db object
serial	Erlaubt das Lesen und Schreiben von Strings über die RS232-Schnittstelle des **KNX**CONTROL-Gerätes (oder über kompatible USB-RS232 Schnittstellen)	
modbus	Erlaubt die Verwaltung ein oder mehrerer MODBUS SLAVE –Geräte über TCP/UDP (Lesen und Schreiben von Werten)	

2.4 INPUTS UND OUTPUTS

Wird ein RUN-SCRIPT ausgeführt, wird ihm immer ein INPUT-Wert übergeben:

- Bei direkter Bedienung des RUN-SCRIPTS über eine Seite der Visualisierung hängt der Wert vom Benutzer ab (z.B.: Eingabe eines Wertes über ein Textfeld, Bedienung eines Buttons, … - abhängig vom grafischen Aspekt, der dem RUN-SCRIPT zugewiesen wurde)
- Bei Ausführung des RUN-SCRIPTS durch die Status-Veränderung eines Objekts, welches als PARENT-Objekt (über ein PASSIVES EREIGNIS) des RUN-SCRIPTS definiert worden ist, wird der Wert des Objekts selbst übernommen, wie in den Einstellungen des RUN-SCRIPTS definiert; dabei können folgende Werte übergeben werden:
  - Den Wert, den das PARENT-Objekt selbst angenommen hat
  - Einen konstanten Wert, welcher bei jeder Veränderung des PARENT-Objekts gesendet wird

In beiden Fällen kann der INPUT-Wert innerhalb des Scripts über folgende Funktion abgerufen werden:

```
input()
```

Beispiel:

```
$val = input();
```

In Abhängigkeit vom PARENT-Objekt, durch welches das RUN-SCRIPT gestartet wurde, kann der INPUT-Wert numerisch oder eine Zeichenkette sein; die PHP-Sprache sieht hier keine besonderen Unterscheidungen vor, da die Variable, welcher der INPUT-Wert zugewiesen wird, automatisch angepasst wird. Natürlich kann der Autor des Scripts selbstständig den Typ des Eingangswerts abfragen und für eigene Kontrollen verwenden. Um z.B. zu überprüfen, ob der Eingangswert eine Zeichenkette ist, kann folgende PHP-Funktion verwendet werden:

```
is_string(input())
```

Die Überprüfung auf Ganzzahlen (Integer) oder Kommazahlen (Float) kann auf folgende Art und Weise erfolgen:

```
is_int(input())
is_float(input())
```

Zu guter Letzt kann noch überprüft werden, ob der Eingabe-Wert eine Zeichenkette mit numerischen Zeichen ist:

```
is_numeric(input())
```

Weitere Details zu den von PHP unterstützten Datentypen sowie die entsprechenden Kontroll- bzw. Umwandlungs-Funktionen können über folgende Dokumentation im Internet bezogen werden:

```
http://www.php.net/manual/en/ref.var.php
```

Nach erfolgter Ausführung gibt das Script einen OUTPUT-Wert zurück; dieser wird dem RUN-Script als Ergebnis übergeben und kann z.B. in der Visualisierung dargestellt werden oder zur Ausführung weiterer Ereignisse genutzt werden. Wurde im RUN-SCRIPT zumindest ein AKTIVES EREIGNIS definiert, wird der Wert zudem den zugewiesenen CHILD-Objekten weitergegeben.

Um einen Wert nach Ausführung des Scripts zurückzugeben, kann folgende Funktion verwendet werden:

```
output()
```

Beispiel:

```
output(1);
```

Der Aufruf der Funktion **output()** beendet die Ausführung des Scripts; jeglicher Code, der in der Script-Datei nach diesem Aufruf platziert ist, wird ignoriert.

Soll zum Abschluss des Scripts kein Wert zurückgegeben werden, reicht es, die Funktion ohne Parameter aufzurufen:

```
output();
```

In diesem Fall:

- wird dem RUN-SCRIPT kein Wert zurückgegeben
- werden die verknüpften AKTIVEN EREIGNISSE nicht ausgeführt

Dass bei Aufruf der Funktion output() ohne Parameter keine AKTIVEN EREIGNISSE ausgeführt werden, darf nicht als Limit, sondern als Erhöhung der Flexibilität gesehen werden. So ist es nämlich möglich, die Ereignisse bereits innerhalb des Scripts zu starten und dabei UNTERSCHIEDLICHE Werte zu übergeben, was bei Verwendung des OUTPUT-Wertes (für alle gleich) nicht möglich wäre. Weitere Details zur Realisierung dieser Funktionalität finden Sie im Abschnitt zur "**surrounding**" - Bibliothek.

2.5 DEBUG, INFO UND FEHLER

Innerhalb des Scripts können nach Belieben Meldungen vom Typ DEBUG (zur Kontrolle der korrekten Funktion des Scripts während dessen Erstellung), INFO (zur Ausgabe von Informationen während der Ausführung des Scripts) oder FEHLER (zur Benachrichtigung bei Fehlern, welche einen korrekten Abschluss des Scripts verhindern) eingefügt werden.

Um diese Meldungen einzubinden, können folgende Befehle verwendet werden:

```
debug()
info()
error()
```

Beispiele:

```
debug("Dies ist eine Debug-Meldung");
info("Dies ist eine Info-Meldung");
error("Dies ist eine Fehler-Meldung");
```

Die Funktion **error**() sieht einen 2. Parameter vor, den sog. RETURN-Code des Scripts selbst. Dieser Code (nicht zu verwechseln mit dem OUTPUT-Wert des Scripts) ist ein numerischer Wert, welcher dem Script-Motor mitteilt, ob das ausgeführte Script korrekt beendet werden konnte. Im Normalfall beträgt dieser Wert immer 0 (Null), was einer fehlerfreien Ausführung entspricht. Wird ein Script durch den Aufruf der Funktion **error**() unterbrochen, kann dieser Code zur Spezifizierung des Fehlers genutzt werden:

```
error("Dies ist eine Fehler-Meldung",1);
```

Ihr **KNX**CONTROL-Gerät sieht einige RETURN-Codes vor, welche über eine statische Konstante genutzt werden können:

RETURN Code	Numerischer Wert	Beschreibung
_DPAD_RESULT_NOERROR	0	Kein Fehler
_DPAD_RESULT_DBERROR	1	Datenbank-Fehler
_DPAD_RESULT_SYSERROR	4	Allgemeiner System-Fehler
_DPAD_RESULT_CONFIGERROR	5	Konfigurations-Fehler
_DPAD_RESULT_USERERROR	6	Benutzer-Fehler (z.B.: durch Eingabe falscher Werte)
_DPAD_RESULT_IOERROR	7	Dateisystem-Fehler (Flash-Speicher des Produkts, z.B. Speicher-Fehler oder Fehler beim Zugriff auf Dateien)
_DPAD_RESULT_TIMEOUT	12	Timeout-Fehler (Script benötigt zu lange)

Die Konstanten können wie folgt in die Fehler-Meldung eingebunden werden:

```
error("Dies ist eine Fehler-Meldung",_DPAD_RESULT_USERERROR);
```

Die Meldungen können beim Testen des Scripts (wird ausführlich im Abschnitt "Debug" erklärt) verwendet werden. Bei der effektiven Verwendung des Scripts (bzw. des Run-Scripts) werden diese Meldungen nicht angezeigt, jedoch im internen Log zur späteren Konsultierung abgelegt.

2.6 EINBINDUNG VON SCRIPTS

Es ist auch möglich, andere SCRIPTS innerhalb eines SCRIPTS einzubinden; dies ermöglicht es, zuvor realisierte SCRIPTS wie eine „persönliche Bibliothek" zu verwenden.

Um ein SCRIPT einzubinden, reicht der Aufruf folgender Funktion:

```
include_script($NameScript);
```

$NameScript ist dabei der Name des einzubindenden Scripts, z.B.:

```
include_script("MeinScript");
```

Auf diesem Weg kann dem eingebundenen Script sogar ein INPUT-Wert übergeben werden, welcher anstelle des Wertes der verknüpften Objekte verwendet wird. In diesem Fall den Wert einfach als 2. Parameter im Aufruf übergeben:

```
include_script("MeinScript","Neuer INPUT-Wert");
```

Diese Funktion gibt den OUTPUT-Wert des eingebundenen Scripts retour. Dieser Wert kann in einer Variablen gespeichert werden und im Anschluss im Haupt-Script weiterverwendet werden:

```
$value = include_script("MeinScript");
```

Als Beispiel sei hier das Script "Verdoppeln", aufgeführt, welches einfach den INPUT-Wert verdoppelt und zurückgibt:

```
$double = input() * 2;
output($double);
```

SCRIPT "Verdoppeln"

Soll dieses Script nun als "Bibliothek" in einem weiteren Script verwendet werden (u.a. auch zur "Verdoppelung" anderer Werte), kann dies wie folgt realisiert werden:

```
$double1 = include_script("Verdoppeln");   //Verdoppelt den INPUT-Wert
$double2 = include_script("Verdoppeln ",2); //Verdoppelt den Wert 2 (gibt 4 zurück)
$double3 = include_script("Verdoppeln ",5); // Verdoppelt den Wert 5 (gibt 10 zurück)
[...]
```

Die INPUT- und OUTPUT-Werte eines Scripts, welches über die Funktion **include_script()** eingebunden wurde, verändern die INPUT- und OUTPUT-Werte des Haupt-Scripts NICHT.

Um eines der Beispiel-Scripts anstelle eines eigenen Scripts einzubinden, bitte einfach die folgende Funktion verwenden (identisch wie **include_script()**):

```
include_sample("BeispielScript");
```

3 SCRIPTS

3.1 EINLEITUNG

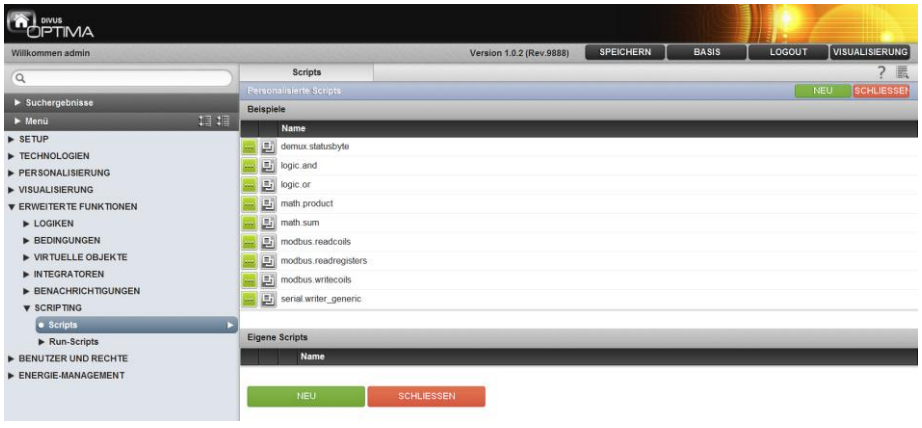
Dieses Kapitel erklärt, wie Sie personalisierte Scripts erstellen, bearbeiten, löschen und exportieren können.

3.2 VERWALTUNGSFENSTER DER SCRIPTS

Um auf das Verwaltungsfenster der Scripts zugreifen zu können, gehen Sie bitte wie folgt vor:

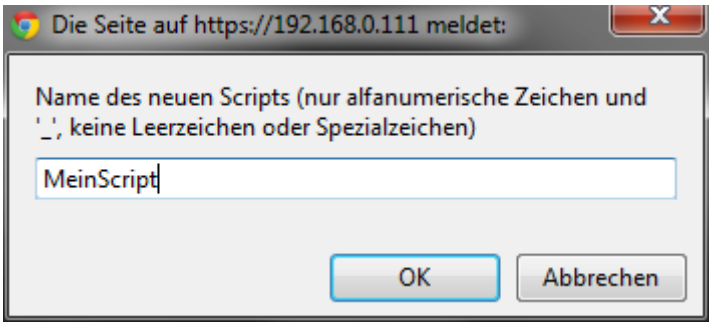
- Loggen Sie Sich als Admin-Benutzer in der OPTIMA-Oberfläche ein
- Wechseln Sie in die Konfigurations-Ebene Ihres **KNX**CONTROL-Gerätes
- Im Navigationsmenü „ERWEITERTE FUNKTIONEN“ → „SCRIPTING“ auswählen
- “SCRIPTS” auswählen

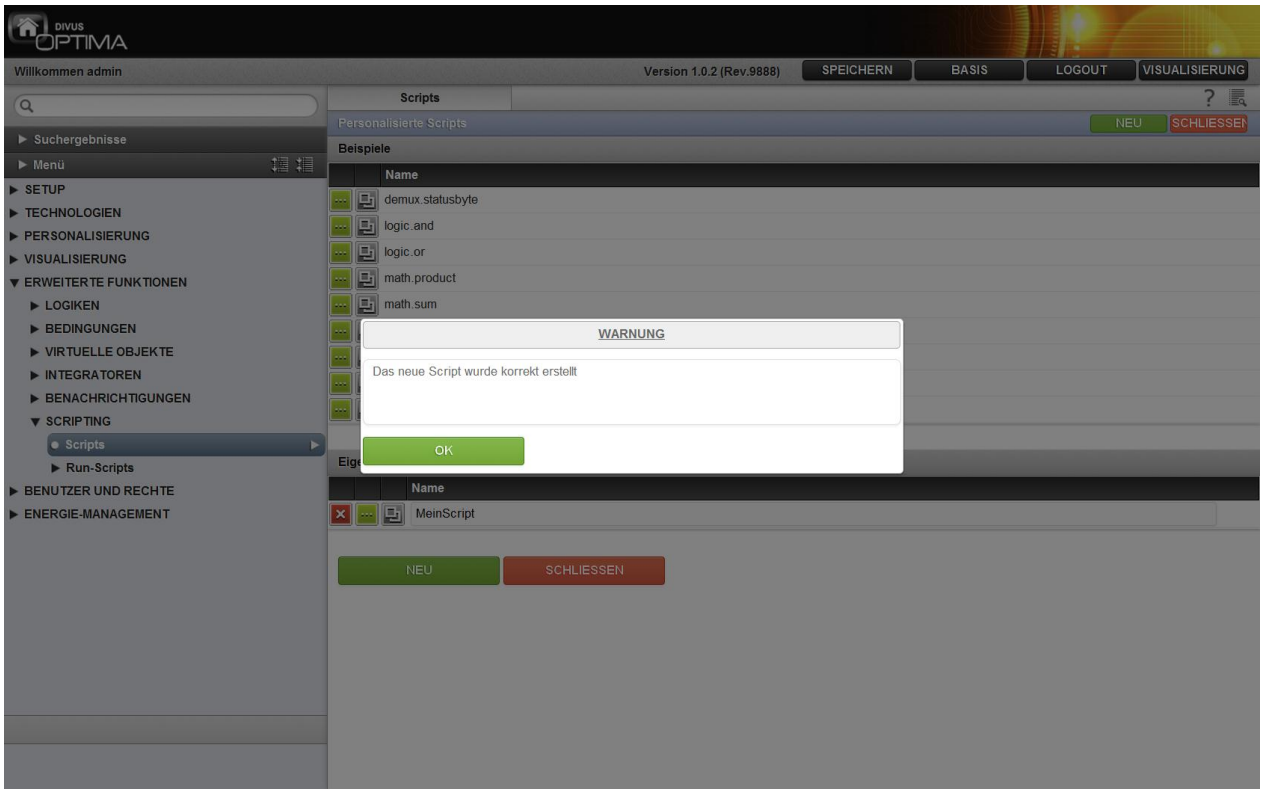
Es wird eine Seite ähnlich jener im Screenshot angezeigt:



Der obere Bereich der Seite zeigt die verschiedenen Beispiel-Scripts, welche in Ihrem **KNX**CONTROL-Gerät vorinstalliert sind. Diese können nicht gelöscht oder bearbeitet werden, sehr wohl aber kopiert werden (um daraus eigene Scripts abzuleiten).

Der untere Bereich hingegen zeigt die selbst erstellten Scripts. Um ein neues Script zu erstellen, reicht ein Klick auf den NEU – Button am Ende der Seite. Nach Eingabe einer Bezeichnung für das neue Script wird dieses angelegt, was durch eine Meldung bestätigt wird:


Die Seite auf https://192.168.0.111 meldet:
Name des neuen Scripts (nur alfanumerische Zeichen und '_', keine Leerzeichen oder Spezialzeichen)
MeinScript
OK
Abbrechen



Will man bei der Erstellung eines Scripts von einem der Beispiel-Scripts starten, muss lediglich der entsprechende KLONEN – Button (grau) links neben der Bezeichnung des Beispiel-Scripts betätigt werden. Auch in diesem Fall muss ein Name für die Kopie eingetragen werden.

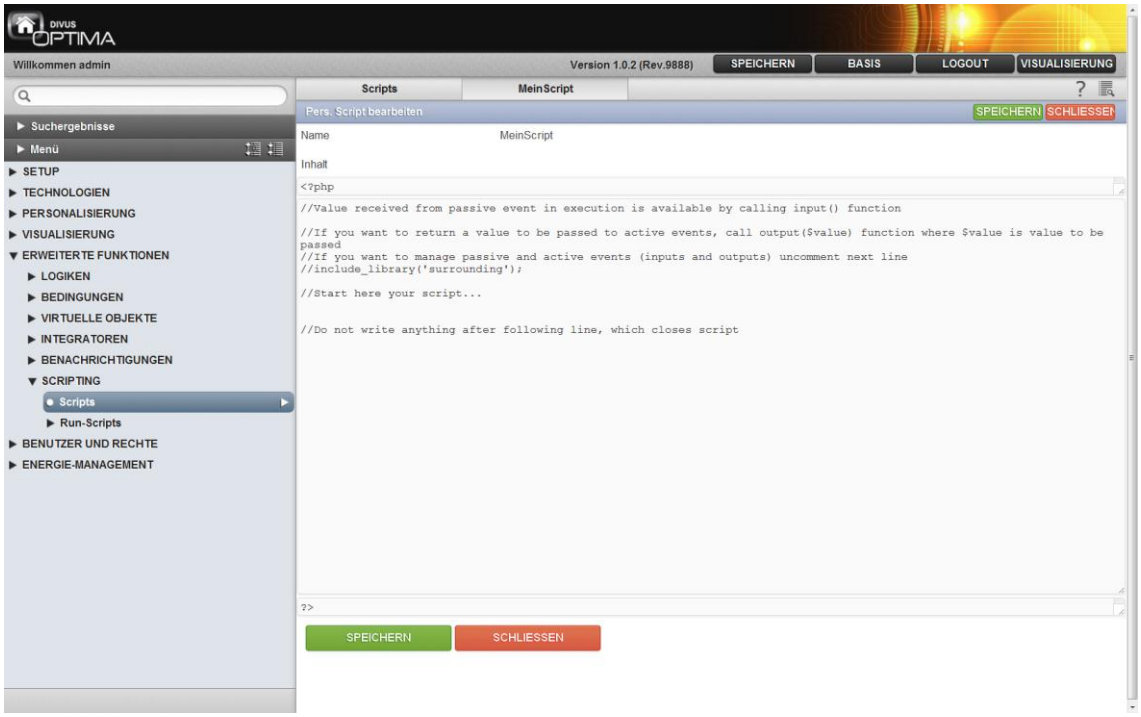
Um ein Script zu löschen, kann einfach auf den entsprechenden LÖSCHEN – Button (rot) geklickt werden. Nach Bestätigung der Aktion wird das Script vom Flash-Speicher des **KNX**CONTROL-Gerätes entfernt (kann nicht rückgängig gemacht werden).

Um hingegen auf das Eigenschaftsfenster eines Scripts zuzugreifen, klicken Sie bitte einfach auf den entsprechenden BEARBEITEN - Button (grün).

3.3 DARSTELLUNG / BEARBEITUNG EINES SCRIPTS

Wird der BEARBEITEN – Button eines eigenen Scripts oder eines Beispiel-Scripts betätigt, so öffnet sich das Eigenschaftsfenster des Scripts, in welchem der enthaltene PHP-Code dargestellt wird. Im Fall von Beispiel-Scripts kann dieser Code nicht bearbeitet werden; bei einem neuen Script hingegen ist der Inhalt editierbar und besteht aus einigen Kommentaren (erkennbar am Präfix "//"), welche einige hilfreiche Informationen zur Script-Erstellung beinhalten, welche in diesem Handbuch vertieft werden.

Folgender Screenshot zeigt das Eigenschaftsfenster eines neuen Scripts:



Wie man erkennen kann, sind die TAGS zum Beginnen des Scripts...

```
<?php
```

... sowie zum Abschließen des Scripts...

```
?>
```

... bereits eingefügt (außerhalb des editierbaren Bereichs) und müssen nicht mehr eingebunden werden! Die folgenden Kapitel enthalten genauere Informationen zur Erstellung von Scripts unter Zuhilfenahme der zur Verfügung stehenden Bibliotheken.

Nach erfolgter Bearbeitung des Codes können die Änderungen über den SPEICHERN – Button übernommen werden. Wird hingegen direkt der SCHLIESSEN – Button betätigt, gehen alle nicht gespeicherten Veränderungen verloren.

3.4 BEISPIEL

Als Beispiel-Code wird im weiteren Verlauf des Handbuchs auf folgenden Code verwiesen (wenn nicht anders erwähnt), welcher den erhaltenen INPUT-Wert unverändert als OUTPUT-Wert zurückgibt, nachdem dieser in einer DEBUG-Meldung angezeigt worden ist (die DEBUG-Meldung ist nur beim Testen des Scripts sichtbar, wie noch genauer erklärt wird):

```
$value = input();
debug("Erhaltener INPUT-Wert: " . $value);
output($value);
```

4 RUN-SCRIPTS

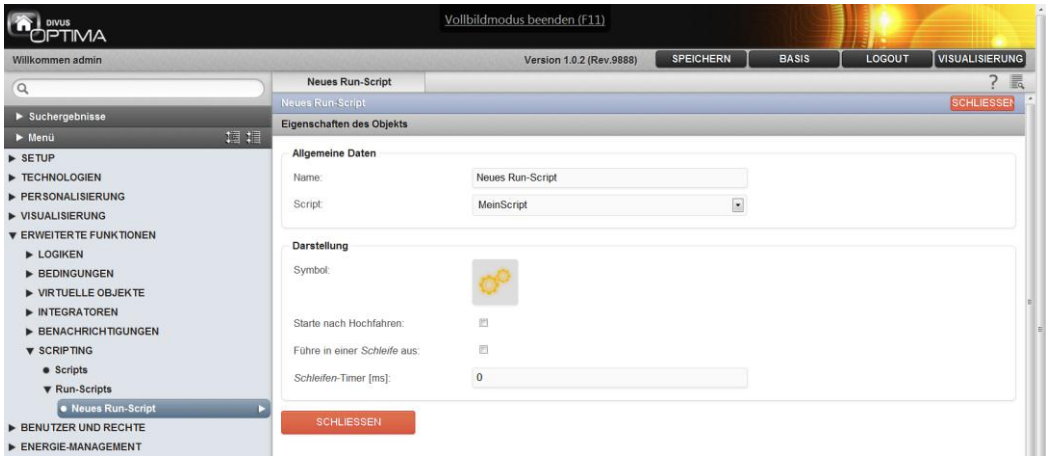
4.1 EINLEITUNG

Dieses Kapitel zeigt, wie Objekte vom Typ RUN-SCRIPT angelegt und verwaltet werden; über diese erfolgt die Ausführung der SCRIPTS innerhalb der Visualisierung.

4.2 ERSTELLUNG EINES RUN-SCRIPTS

Um ein neues Run-Script zu erstellen, gehen Sie bitte wie folgt vor:

- Loggen Sie Sich als Admin-Benutzer in die OPTIMA-Oberfläche ein und wechseln in die Konfigurations-Ebene
- Im Navigationsmenü „ERWEITERTE FUNKTIONEN“ → „SCRIPTING“ → „RUN-SCRIPTS“ auswählen
- Den „HINZUFÜGEN“-Button in der Toolbar betätigen
- Das Eigenschaftsfenster des neuen Run-Scripts über die „BEARBEITEN“- Funktion öffnen



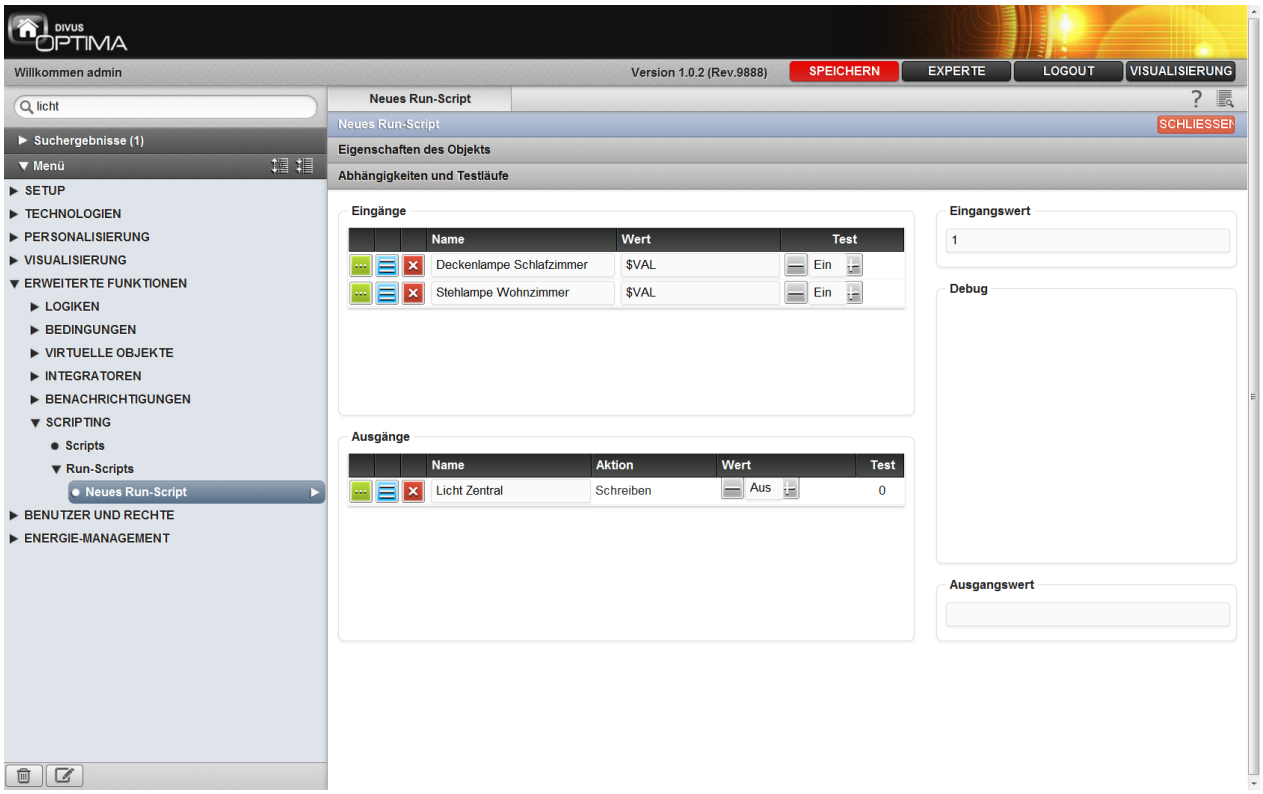
Neben der Bezeichnung des neuen Run-Scripts muss auch das Script definiert werden (über das Drop-Down-Menü), welches ausgeführt werden soll. Des Weiteren kann der grafische Aspekt des Run-Scripts, welcher die Darstellung innerhalb der Seiten der Visualisierung sowie die Interaktionsmöglichkeiten des Benutzers mit dem Run-Script bestimmt, angepasst werden.

Zudem können folgende Optionen für das Run-Script definiert werden:

STARTE NACH HOCHFAHREN	Bei Aktivierung dieser Checkbox wird das Run-Script direkt nach dem Start des **KNX**CONTROL-Gerätes ausgeführt
FÜHRE IN EINER SCHLEIFE AUS	Bei Aktivierung dieser Checkbox wird das Run-Script kontinuierlich in einer Schleife ausgeführt, bis es entweder manuell oder via Ereignis gestoppt wird. Details hierzu finden Sie im Abschnitt „Ausführung im Hintergrund"
SCHLEIFEN-TIMER	Bei Aktivierung der Ausführung in einer Schleife, geben Sie hier bitte die Dauer eines Schleifendurchgangs an.

Ab Version 1.1 ist es zudem möglich, ein Run-Script manuell über die entsprechenden Schaltflächen zu starten und zu stoppen.

Der untere Teil des Eigenschaftsfensters ("ABHÄNGIGKEITEN UND TESTLÄUFE") erlaubt die Verwaltung der Verknüpfungen des Run-Scripts mit anderen Objekten sowie das Testen (DEBUG) des Scripts in Echtzeit.



4.2.1 EINGÄNGE

Der Abschnitt **EINGÄNGE** erlaubt das Verknüpfen ein oder mehrerer Objekte, bei deren Status-Änderung das Run-Script ausgeführt werden soll. Bezieht man dies auf andere Objekte der Software, so kann man diese Objekte als "PASSIVE EREIGNISSE" bezeichnen, welche die Ausführung des Run-Scripts (und somit des verknüpften Scripts) bestimmen.

Nachdem ein Objekt in diesem Bereich verknüpft worden ist, muss definiert werden, welcher Wert dem Run-Script bei Status-Änderungen des Objekts übergeben werden soll. In Unterschied zu anderen Objekten der Software, bei welchen dieser Wert über ein Drop-Down-Menü ausgewählt werden kann, muss der Wert hier manuell eingegeben werden, da nur dadurch die maximale Flexibilität bei der Verkettung von mehreren Run-Scripts gewährleistet werden kann. Dieses Feld kann folgende Werte beinhalten:

$VAL	Der aktuelle Wert des Objekts wird dem Run-Script als INPUT-Wert übergeben
$NVAL	Der invertierte Wert des Objekts wird dem Run-Script als INPUT-Wert übergeben
Eine beliebige Nummer / beliebige Zeichenkette	Der eingetragene, konstante Wert wird dem Run-Script als INPUT-Wert übergeben, unabhängig vom Wert des Objekts selbst

In jedem der Fälle kann der INPUT-Wert innerhalb des Scripts über den Befehl…

```
input()
```

…abgerufen werden, wie bereits in den vorhergehenden Kapiteln erwähnt.

Der Wert innerhalb dieses Feldes ist nicht die einzige Möglichkeit, innerhalb des Scripts auf den Wert eines verknüpften Objekts zuzugreifen. Über die „surrounding" – Bibliothek kann, wie im weiteren Verlauf des Handbuchs beschrieben wird, auf den Status jedes einzelnen PARENT-Objekts zugegriffen werden. Das Wert-Feld ist in diesem Moment nur für den INPUT-Wert des Scripts wichtig, welcher über den Befehl **input()** ausgelesen werden kann, wie bereits gesehen.

Ab Version 1.1 kann jedem Objekt ein sogenannter **Identifier** (bestehend aus einem Text-String ohne Leer- oder Spezialzeichen) zugewiesen werden, welcher die Verwendung des Objekts innerhalb eines Scripts der SURROUNDING-Bibliothek erleichtert. Wie im nächsten Kapitel noch genauer erklärt wird, kann über folgenden Befehl das EINGANGS-Objekt innerhalb des Scripts über dessen Identifier aufgerufen werden:

```
$me->getParentByIdentifier("Identifier")
```

Die Spalte **Variable** zeigt den Namen der Variablen, über welchen das EINGANGS-Objekt innerhalb des Scripts verfügbar ist.

Die Spalte "Variable" kann nicht bei allen Aktionen (Objekte verschieben, hinzufügen, neu ordnen) korrekt aktualisiert werden. Bitte verwenden Sie deshalb den Button "AKTUALISIEREN" nach Durchführung zuvor besagter Aktionen, um alle Variablen korrekt anzuzeigen.

Die Spalte **Test** erlaubt die simulierte Ausführung des Run-Scripts durch Einstellung des gewünschten Wertes (über das Drop-Down-Menü) eines der PARENT-Objekte. Die im Drop-Down-Menü verfügbaren Werte hängen vom verknüpften Objekt-Typ ab.

4.2.2 AUSGÄNGE

Auf dieselbe Art und Weise wie die Eingänge können auch die **AUSGÄNGE** über entsprechenden Abschnitt definiert werden, welche nach Abschluss des Scripts angesteuert werden können. Für jedes Objekt in diesem Abschnitt kann die auszuführende Aktion bestimmt werden und – wenn vom Objekt unterstützt – der zu setzende Wert. Als Wert kann auch der "WERT DES RUN-SCRIPTS" übergeben werden. In diesem Fall wird dem Objekt der OUTPUT-Wert des Scripts übergeben, welcher über folgenden Befehl bestimmt wird:

```
output(...);
```

Wie bereits erwähnt, kann die Ausführung von Aktionen auf den als Ausgänge definierten Objekten verhindert werden, wenn der Funktion **output()** kein Parameter übergeben wird. In diesem Moment muss sich das Script selbst um die Ausführung ev. gewünschter Aktionen kümmern, wie im Kapitel zur "**surrounding"** – Bibliothek näher erklärt wird.

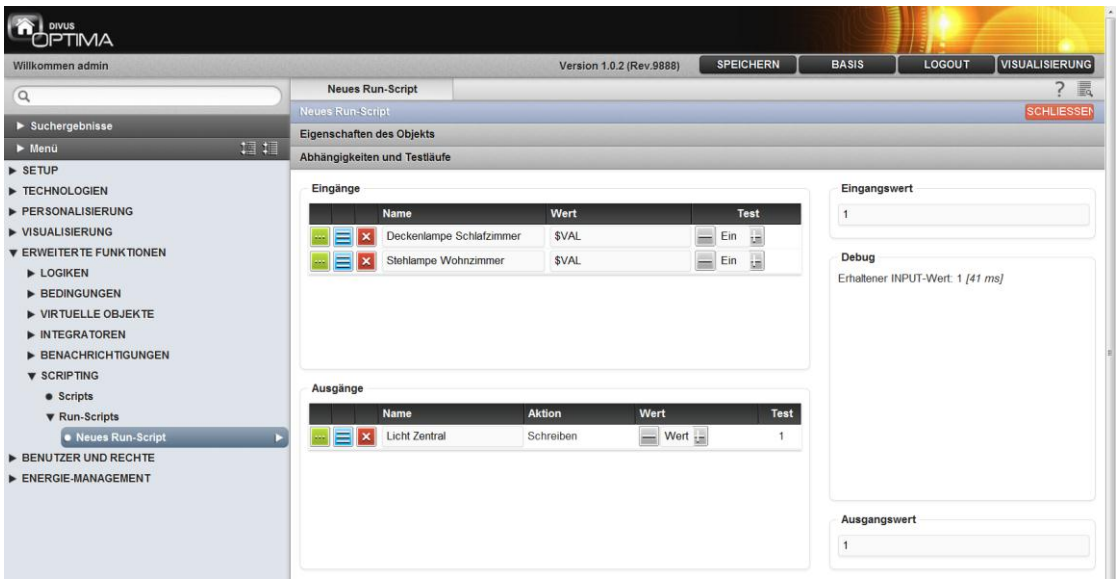
Die Spalte **Test** (zu Beginn leer) zeigt den Status eines jeden als Ausgang definierten Objekts nach Ausführung des Run-Scripts an, natürlich in Abhängigkeit von den übergebenen INPUT-Werten.

4.3 DEBUG

Nachdem die gewünschten Objekte als Eingänge / Ausgänge des Run-Scripts[1] definiert worden sind, kann die Ausführung des Scripts dank der "DEBUG" – Funktion getestet werden; diese Funktion führt das verknüpfte Script aus (alle Aktionen innerhalb des Scripts werden zu 100% ausgeführt), ignoriert jedoch die Ausführung der Aktionen verknüpften Ausgangs-Objekte (CHILD-Objekte), deren Status lediglich über die Spalte "TEST" angezeigt, jedoch nicht gesetzt wird.

Um eine Status-Änderung eines Objektes im Eingang zu simulieren, kann einfach der entsprechende Wert in der Spalte "TEST" über das Drop-Down-Menü verändert werden. Der gewählte Wert wird automatisch ins Feld "EINGANGSWERT" geschrieben und dem Script als INPUT-Wert übergeben.

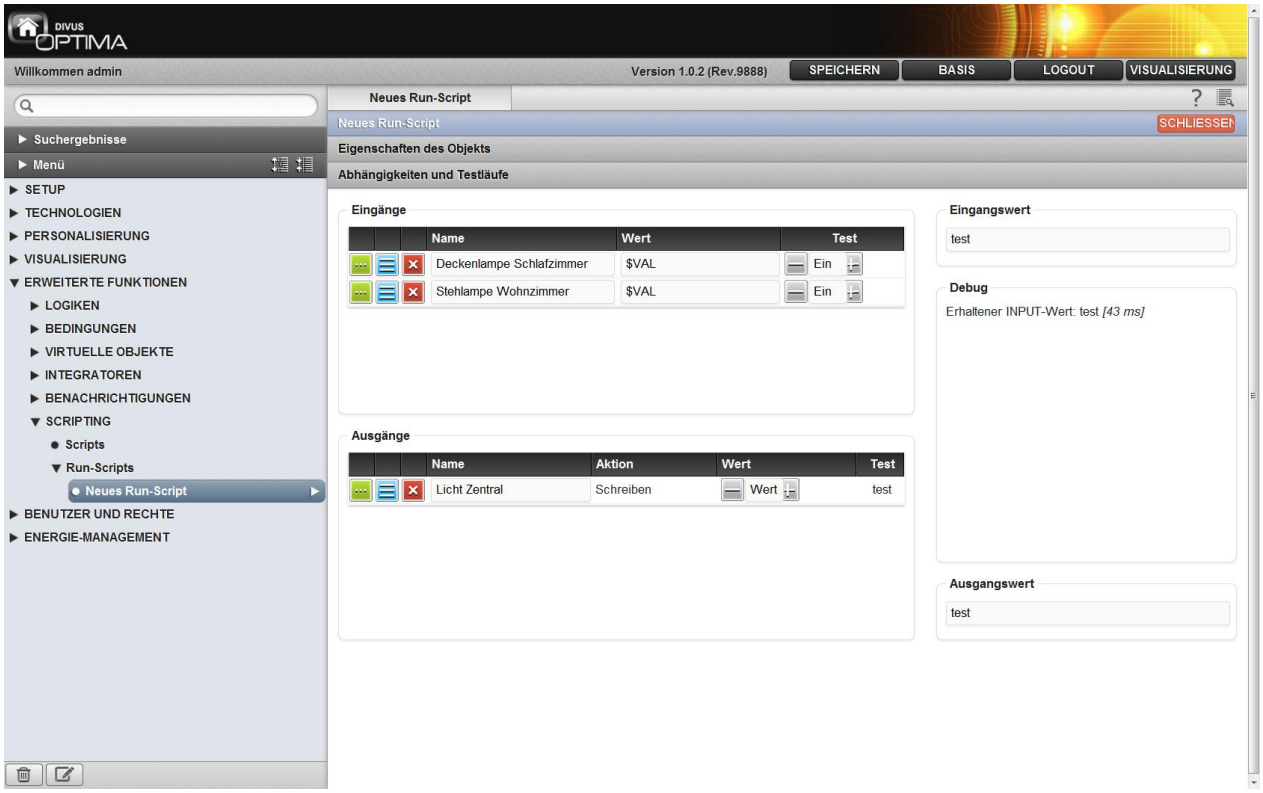
Sind im Script Debug- oder andere Meldungen enthalten, so werden diese im Fenster "DEBUG" angezeigt. Dieses Fenster zeigt im Prinzip alle Ausgaben des Scripts an, also auch jene, die über weitere PHP-Funktionen (wie z.B.: **print()**, **echo()**, **print_r()** usw.) eingebunden worden sind:



[1] Um ein Run-Script auszuführen, müssen noch keine Ein- bzw. Ausgänge verknüpft sein; in diesem Fall kann die Simulation direkt über das Feld "EINGANGSWERT" gestartet werden.

Der von der Funktion **output**() zurückgegebene Wert wird im Feld "AUSGANGSWERT" angezeigt; des Weiteren wird die Spalte "TEST" der Objekte im Ausgang je nach Konfiguration verändert.

Alternativ können Werte dem Script auch manuell übergeben werden. In diesem Fall schreiben Sie den Wert einfach direkt ins Feld "EINGANGSWERT" und drücken die Taste "TAB" auf Ihrem Keyboard (oder alternativ die Taste "ENTER" oder, noch einfacher, klicken Sie einfach eine leere Stelle außerhalb des Eingabefeldes an):



Entspricht der EINGANGSWERT nicht dem Wert, welcher in der Spalte TEST eines Objektes im Eingang gesetzt wurde, bzw. entspricht der Wert in der Spalte TEST eines Objektes im Ausgang nicht dem AUSGANGSWERT des Scripts, überprüfen Sie bitte die Richtigkeit der jeweiligen Einträge in der Spalte WERT.

Dieser Simulationsmodus dient nicht nur der Überprüfung des Run-Scripts sowie dessen Verknüpfungen, sondern auch der Überprüfung des Script-Codes selbst; zu diesem Zweck können vor allem eventuelle im Script eingefügte DEBUG-Meldungen sehr hilfreich sein.

4.4 AUSFÜHRUNG IM HINTERGRUND

Wie bereits angedeutet, können Run-Scripts auch im Hintergrund ausgeführt werden, um z.B. wiederholt Aktionen durchzuführen (z.B. das periodische Abfragen eines Wertes), ohne dass eine direkte Interaktion von Seiten des Benutzers erforderlich ist.

Über die erste Option kann definiert werden, ob das Run-Script nach dem Start der Software automatisch ausgeführt werden soll. Bei Aktivierung dieser Option wird bei jedem Start der Software (z.B. nach einem Neustart des **KNX**CONTROL-Gerätes) das Run-Script mitgestartet.

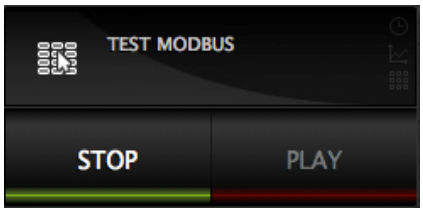
Wird die Option "FÜHRE IN EINER SCHLEIFE AUS" aktiviert, wird das Script automatisch so lange wiederholt ausgeführt, bis es entweder manuell oder über ein entsprechendes EREIGNIS gestoppt wird. Kombiniert man diese Option mit der ersten, so kann z.B. eine kontinuierliche Ausführung eines Scripts vom Start der Visualisierung an realisiert werden, welches im Hintergrund ohne Notwendigkeit von Benutzer-Eingaben läuft.

Wird die Ausführung in einer Schleife aktiviert, muss auch ein Schleifen-Timer für die einzelnen Wiederholungen in Millisekunden angegeben werden. Der Scripting-Motor berechnet automatisch die Zeit, die zur Ausführung des Scripts selbst benötigt wird und zieht diese vom eingetragenen Wert ab, sodass jeder Schleifenzyklus exakt (mit einer minimalen Toleranz) dem eingetragenen Timer-Wert entspricht.

Bei Aktivierung der Ausführung in einer Schleife wird der AUSGANGSWERT des Scripts (von der Funktion **output()** generiert) automatisch der EINGANGSWERT für die nächste Ausführung. Auf diese Art und Weise kann das Script die Werte inkrementell übergeben; sollte das nicht gewünscht sein, rufen Sie im Script bitte NICHT die Funktionen **input()** und **output()** auf!

4.5 DARSTELLUNG IN DER VISUALISIERUNG

Wird ein Run-Script in eine Seite der Visualisierung eingefügt, so entspricht die Darstellung jener im folgenden Screenshot:



Bei Betätigung des PLAY – Buttons wird das Run-Script ausgeführt; das Symbol verändert sich dementsprechend für die Dauer der Ausführung des Scripts. Wird das Script in einer Schleife ausgeführt, so bleibt das Objekt in Ausführung, bis der STOP – Button gedrückt wird oder das System heruntergefahren wird.

Das Unterbrechen eines Run-Scripts, welches in einer Schleife ausgeführt wird, erfolgt erst nach Ablauf des Script-Codes, also in der Wartezeit bis zur Ausführung des nächsten Schleifen-Zyklus. Es kann somit passieren, dass das Run-Script auch nach Betätigung des STOP – Buttons noch eine Weile in Ausführung bleibt, bevor es unterbrochen wird (in Abhängigkeit von der Zeit, die das Script selbst zum Durchlaufen benötigt).

5 "OBJECT" - BIBLIOTHEK

5.1 EINLEITUNG

Die Bibliothek "object" erlaubt die Verwaltung von ein oder mehreren Objekten der Visualisierung Ihres **KNX**CONTROL-Gerätes innerhalb eines Scripts, in der Form von Variablen. Diese Variablen sind zu 100% PHP-Objekte und stellen dementsprechend verschiedene Attribute und Methoden zur Verfügung, über welche mit der Software interagiert werden kann. Es ist z.B. möglich, in Echtzeit den Status eines jeden Objekts der Visualisierung auszulesen oder Aktionen auszuführen (wie z.B. die Bedienung von KNX-Funktionen, die Ausführung von Szenarien usw.).

5.2 EINBINDUNG DER BIBLIOTHEK

Um die Bibliothek in ein Script einzubinden, reicht folgender Aufruf zu Beginn des Scripts:

```
include_library("object");
```

Von der nächsten Zeile an kann im Script folgende Klasse verwendet werden:

```
objM
```

Diese Klasse ("Object Manager" bzw. "Objekt-Verwalter") stellt diverse Funktionen (bzw. "Methoden") zur Verfügung, um Objekte der Software zu laden und Operationen auszuführen. Die Klasse objM wird über "statische Aufrufe" verwendet, wie z.B.:

```
objM::objGet(...);
```

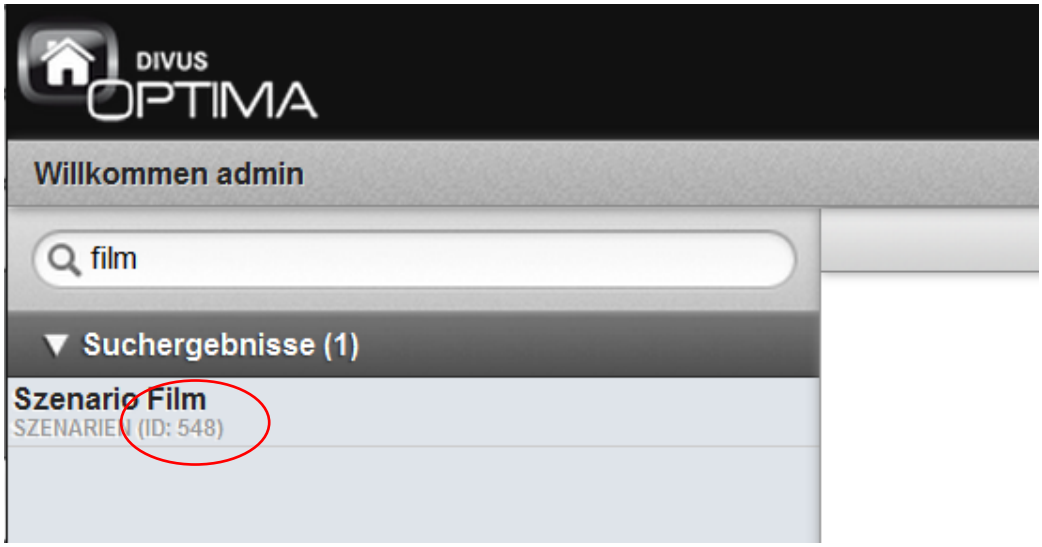
Wie gezeigt, erfolgen die statischen Aufrufe beginnend mit dem Namen der Klasse (ohne das typische Variablen-Präfix "$"), gefolgt von zwei Doppelpunkten und dem Namen der aufzurufenden Funktion; diese Aufrufe unterscheiden sich von normalen Objekt-Aufrufen (deren Name immer das Präfix "$" aufweist), welche das Trennzeichen "->" vor dem Namen der aufzurufenden Funktion benötigen.

5.3 LADEN EINES OBJEKTS

Um ein Objekt als Variable in ein Script zu laden, kann folgender Aufruf verwendet werden:

```
$namevariable = objM::objGet(XXX);
```

XXX ist die einzigartige numerische ID, welche das Objekt innerhalb der Software identifiziert. Die ID eines Objekts kann über die Suchfunktion in der Konfigurations-Ebene festgestellt werden, wie in folgendem Ausschnitt ersichtlich:



Für das im Screenshot gezeigte "Szenario Film" würde der Aufruf im Script z.B. folgendermaßen lauten:

```
$scenarioFilm = objM::objGet(548);
```

Die Variable, die nun eine Referenz des gewählten Objekts enthält, stellt folgende Eigenschaften im Script zur Verfügung:

Eigenschaft	Beschreibung	Beispiel Zugriff
id	ID des Objekts	$obj->id
name	Bezeichnung des Objekts innerhalb der Visualisierung	$obj->name
state	Aktueller Status des Objekts. HINWEIS: diese Eigenschaft kann nur in Kombination mit Szenarien verwendet werden; in diesem Fall: 1 → Szenario gestoppt 2 → Szenario in Ausführung	$obj->state
value	Aktueller Wert des Objekts HINWEIS: diese Eigenschaft kann mit allen Objekt-Typen, die über einen Wert verfügen, verwendet werden, außer den Szenarien (siehe Eigenschaft „state")	$obj->value
enabled	Gibt Auskunft, ob das Objekt aktiv ist oder nicht. Mögliche Werte: 0 , 1	$obj->enabled
msp	Wichtigster Parameter des Objekts, in Abhängigkeit des Typs. Einige Beispiele: • KNX Objekte: Gruppenadresse	$obj->msp

	• Logiken: Logischer Ausdruck • Szenarien: Anzahl der Wiederholungen	
type	Funktions-Objekttyp des Objekts, wird vom Kommunikationsdienst verwendet. Einige Beispiele: • KNX Objekte: "EIBOBJECT" • Virtuelle Objekte: "VIRTUALOBJECT" • Szenarien: "SCENARIO" • Logiken: "LOGIC" • Bedingungen: "CONDITION" • Run-Scripts: "CSCMD" *HINWEIS: eine vollständige Liste der verfügbaren Typen finden Sie im Anhang, Kapitel 12.1*	`$obj->type`
phpclass	Grafischer Web-Objekttyp, wird von der Web-Oberfläche verwendet. Einige Beispiele: • KNX Objekte: "dpadEibObject" • Virtuelle Objekte: "dpadVirtualObject" • Szenarien: "dpadScenario" • Logiken: "dpadLogic" • Bedingungen: "dpadCondition" • Run-Scripts: "dpadScriptRunner" *HINWEIS: nicht immer ist die Verbindung zwischen "type" und "phpclass" einzigartig: in einigen Fällen nutzen mehrere "phpclass" denselben "type". Eine vollständige Liste der verfügbaren grafischen Typen finden Sie im Anhang, Kapitel 12.2*	`$obj->phpclass`
values_type	Datentyp oder Kodierung. *HINWEIS: hängt vom Objekttyp ab*	`$obj->values_type`
options	Array mit optionalen Werten. *HINWEIS: hängt vom Objekttyp ab*	`$obj->options` `$obj->options["..."]`

5.4 VERKNÜPFUNGEN MIT ANDEREN OBJEKTEN

Um Verknüpfungen zwischen einem Objekt und weiteren Objekten, entweder vom Typ "PARENT" ("PASSIVE EREIGNISSE" für das Referenz-Objekt) oder vom Typ "CHILD" ("AKTIVE EREIGNISSE" für das Referenz-Objekt) zu erkennen, stehen folgende Methoden zur Verfügung, welche direkt auf der Variablen mit der Objekt-Referenz aufgerufen werden (und nicht auf der Klasse "objM"):

```
$object->loadParents();
$object->loadChilds();
```

Die beiden Funktionen geben entsprechende Arrays retour:

```
$object->parentRelations[];
$object->childRelations[];
```

Diese Arrays enthalten für jede Verknüpfung mit weiteren Objekten (sowohl "PARENT" als auch "CHILD") ein PHP-Objekt vom Typ "**relation**", welches über folgende Eigenschaften verfügt:

Eigenschaft	Beschreibung	Beispiel Zugriff
id	ID der Verknüpfung innerhalb der Datenbank HINWEIS: diese ID steht für die Verknüpfung selbst und darf nicht mit der Objekt-ID der verknüpften Objekte („PARENT" bzw. "CHILD", siehe unten) verwechselt werden	`$relation->id`
parent child	Referenz auf die Objekte, die entweder als "PARENT" oder "CHILD" verknüpft sind HINWEIS: diese Objekte sind vollwertige PHP-Objekte mit denselben Eigenschaften wie in Abschnitt 5.3 erklärt	`$relation->parent` `$relation->child`
condition	Im Falle von EREIGNIS-Verknüpfungen (AKTIV oder PASSIV) der Filter auf das PARENT-Objekt, welcher – wenn korrekt erfüllt – die Ausführung des Ereignisses bestimmt. Bei "statischen" oder "strukturierten" Verknüpfungen (z.B. Verknüpfungen eines Objekts mit einer Räumlichkeit) wird dieses Feld nicht verwendet und enthält normalerweise den Ausdruck "NO-CONDITION"	`$relation->condition`
action	Im Falle von EREIGNIS-Verknüpfungen (AKTIV oder PASSIV) die Aktion, die auf dem CHILD-Objekt ausgeführt wird. Bei "statischen" oder "strukturierten" Verknüpfungen (z.B. Verknüpfungen eines Objekts mit einer Räumlichkeit) wird dieses Feld nicht verwendet und enthält normalerweise den Ausdruck "NO-ACTION"	`$relation->action`
value	Im Falle von EREIGNIS-Verknüpfungen (AKTIV oder PASSIV) der Wert, der dem CHILD-Objekt übergeben wird (wenn verfügbar). Bei "statischen" oder "strukturierten" Verknüpfungen (z.B. Verknüpfungen eines Objekts mit einer Räumlichkeit) wird dieses Feld nicht verwendet und enthält normalerweise den Ausdruck "NO-VALUE"	`$relation->value`
enabled	Gibt Auskunft, ob das Objekt aktiv ist oder nicht. Mögliche Werte: 0 , 1	`$relation->enabled`
options	Liste von optionalen Einstellungen der Verknüpfung; diese Einstellungen, wenn vorhanden präsentieren sich in der Form… param1='value1'|param2='value2'|... … also über das Zeichen "|" verkettet. Numerische Werte sind ev. ohne Hochkomma aufgeführt.	`$relation->options`

Oben aufgeführte Eigenschaften spiegeln die Werte der Datenbank-Tabelle DPADD_OBJECT_RELATION wider, welche von der Software zur Verwaltung der Verknüpfungen zwischen Objekten verwendet wird.

Die Funktionen **loadChilds**() und **loadParents**() laden normalerweise nur Verknüpfungen vom Typ EREIGNIS, welche innerhalb von Scripts die meiste Verwendung finden. Um alle Verknüpfungen zu laden, der Funktion einfach den Parameter *true* übergeben:

```
$object->loadChilds(true);
$object->loadParents(true);
```

Sollen hingegen nur spezifische Verknüpfungen geladen werden, einfach *false* als ersten Parameter übergeben sowie, als 2. Parameter, ein Array mit den gewünschten Verknüpfungen:

```
$object->loadChilds(false,["GENERIC_RELATION","ACTION_RELATION"]);
$object->loadParents(false,["GENERIC_RELATION","ACTION_RELATION"]);
```

Man beachte, dass die Eigenschaften "parent" und "child" einer Verknüpfung selbst auch PHP-Objekte sind, mit denselben Eigenschaften wie das Start-Objekt. Um z.B. auf den aktuellen Wert des ersten CHILD-Objekts des Start-Objekts zuzugreifen, kann folgender Ausdruck verwendet werden:

```
$firstChildValue = $object->childRelations[0]->child->value;
```

wobei:

`$object->childRelations[0]`	Erste Verknüpfung mit CHILD-Objekten (Index 0)
`$object->childRelations[0]->child`	Referenz auf das CHILD-Objekt der Verknüpfung[2]
`$object->childRelations[0]->child->value`	Wert des CHILD-Objekts

Um zu verstehen, wie ein Objekt sowie seine Verknüpfungen zu PARENT- und CHILD-Objekten aufgebaut sind, kann folgender Aufruf hilfreich sein (immer vorausgesetzt dass *$object* das Haupt-Objekt ist, mit welchem gearbeitet werden soll):

```
echo "<pre>"; print_r($object); echo "</pre>";
```

Der Befehl **print_r** (eingeschlossen von den HTML-Tags "<pre>" und "</pre>", welche die Ausgabe in ein leserliches Format bringen) zeigt bei Ausführung des Scripts im DEBUG-Fenster (über ein Run-Script, wie in Kapitel 4.3 beschrieben) die gesamte Struktur des Objekts, welches als Parameter übergeben wurde. Es folgt ein Beispiel der Ausgabe dieses Befehls auf einem Objekt vom Typ "Szenario":

```
obj Object
(
    [id] => 548
    [name] => Szenario Film
    [state] => 1
    [value] =>
    [enabled] => 1
    [options] => Array
        (
        )

    [relations] => Array
        (
        )
```

[2] In diesem Fall wäre das PARENT-Objekt der Verknüpfung wieder das Start-Objekt!

```
   [childRelations] => Array
      (
         [0] => relation Object
            (
               [id] => 872
               [parent] => obj Object
*RECURSION*
               [child] => obj Object
                  (
                     [id] => 487
                     [name] => Licht - Wohnzimmer
                     [state] => -1
                     [value] => 0
                     [enabled] => 1
                     [options] => Array
                        (
                           [category] => lighting
                        )

                     [relations] => Array
                        (
                        )

                     [childRelations] => Array
                        (
                        )

                     [childsLoaded] =>
                     [parentRelations] => Array
                        (
                        )

                     [parentsLoaded] =>
                     [msp] => 0/0/4
                  )

               [condition] => NO-CONDITION
               [action] => NO-ACTION
               [value] => 0
               [enabled] => 1
               [options] => Array
                  (
                  )

            )

         [1] => relation Object
            (
               [id] => 873
               [parent] => obj Object
*RECURSION*
               [child] => obj Object
                  (
                     [id] => 491
                     [name] => Licht - Gang
                     [state] => -1
                     [value] => 1
                     [enabled] => 1
                     [options] => Array
                        (
                           [category] => lighting
                        )

                     [relations] => Array
                        (
                        )

                     [childRelations] => Array
                        (
                        )

                     [childsLoaded] =>
                     [parentRelations] => Array
                        (
                        )
```

```
                        [parentsLoaded] =>
                        [msp] => 0/0/6
                    )

                [condition] => NO-CONDITION
                [action] => NO-ACTION
                [value] => 0
                [enabled] => 1
                [options] => Array
                    (
                    )

            )

        [2] => relation Object
            (
                [id] => 875
                [parent] => obj Object
*RECURSION*
                [child] => obj Object
                    (
                        [id] => 503
                        [name] => Licht Dimmer
                        [state] => -1
                        [value] =>  75
                        [enabled] => 1
                        [options] => Array
                            (
                                [category] => lighting
                            )

                        [relations] => Array
                            (
                            )

                        [childRelations] => Array
                            (
                            )

                        [childsLoaded] =>
                        [parentRelations] => Array
                            (
                            )

                        [parentsLoaded] =>
                        [msp] => 0/0/15
                    )

                [condition] => NO-CONDITION
                [action] => NO-ACTION
                [value] => 0
                [enabled] => 1
                [options] => Array
                    (
                    )

            )

    )

    [childsLoaded] => 1
    [parentRelations] => Array
        (
        )

    [parentsLoaded] => 1
)
```

Wie ersichtlich, beinhaltet das Szenario 3 Verknüpfungen mit CHILD-Objekten (festgehalten im Array *childRelations*, über Index 0,1 und 2); jede dieser Verknüpfungen enthält, neben den verschiedenen Eigenschaften, auch eine Referenz auf die CHILD-Objekte, welche ihrerseits über diverse Eigenschaften verfügen.

Man beachte dass die Verknüpfungen auch die Referenz auf das PARENT-Objekt beinhalten; da es sich in diesem Fall dabei jedoch um das Start-Objekt handelt, wird diese Referenz von der Funktion **print_r** mit folgender Bezeichnung herausgehoben:

```
*RECURSION*
```

Der Befehl **print_r**, obwohl sehr hilfreich in der Erstellungs- und Debug-Phase des Scripts (um die Objekt-Struktur bzw. im Allgemeinen die Datenstruktur zu verstehen), ist bei der effektiven Verwendung des Scripts vollkommen nutzlos (wie jede andere Funktion zur Ausgabe von Meldungen) und verlängert unnötig die Ausführdauer des Scripts. Deshalb wird empfohlen, sämtliche Ausgabe-Funktionen nach Abschluss der Debug-Phase zu entfernen.

Ab Version 1.1 kann des Weiteren ein Array der Objekte (die entweder im EINGANG oder im AUSGANG verknüpft sind) über eine Eigenschaft der Verknüpfung mit einem bestimmten anderen Objekt ($object in untenstehenden Beispielen) abgerufen werden. Dies erfolgt über folgende Befehle:

```
$object->getParentsByRelationField($fieldName,$fieldValue);
$object->getChildsByRelationField($fieldName,$fieldValue);
```

Wobei

$fieldName	Name der Eigenschaft der Verknüpfung, welche als Filter verwendet wird
$fieldValue	Wert der Eigenschaft

Will man hingegen anstelle der Objekte die Verknüpfung selbst abrufen, so können folgende Befehle genutzt werden:

```
$object->getParentRelationsByField($fieldName,$fieldValue);
$object->getChildRelationsByField($fieldName,$fieldValue);
```

Die zu übergebenden Argumente sind dieselben wie zuvor.

Zuletzt kann das Objekt (entweder im EINGANG oder im AUSGANG) auch anhand des manuell zugewiesenen **Identifiers** (siehe Kapitel 4.2.1) über folgende Funktionen abgerufen werden:

```
$object->getParentByIdentifier($identifier);
$object->getChildByIdentifier($identifier);
```

5.5 BEDIENUNG EINES OBJEKTS

Der Objekt-Verwalter **objM** erlaubt die Versendung von Befehlen an jegliches Objekt der Software, in derselben Form wie es über die Visualisierungsseiten der OPTIMA-Oberfläche bzw. über Logiken, Szenarien usw. realisiert wird...

Die folgende Funktion zeigt die allgemeine Methode, um Operationen auf Objekten durchzuführen:

```
objM::objPerformOperation(ID,OPERATION,VALUE);
```

wobei

ID	ID des Objekts, auf welchem die Operation durchgeführt werden soll
OPERATION	Durchzuführende Operation
VALUE	Wenn vorgesehen, der Wert, der an das Objekt übergeben wird

Beispiel: um ein KNX-Objekt vom Typ EIN/AUS einzuschalten (wird für das Beispiel mit der ID "*123*" versehen), muss folgende Funktion aufgerufen werden...

```
objM::objPerformOperation(123,"SETVALUE",1);
```

... während zur Ausführung des zuvor gesehenen Szenarios (ID 548) folgender Aufruf ausreicht...

```
objM::objPerformOperation(548,"EXECUTE");
```

... ohne dass ein Wert übergeben wird, da die Ausführung von Szenarien keine Werte vorsieht, wie in Kürze erläutert wird.

Folgende Tabelle zeigt die vorgesehenen Operationen (und ev. deren Werte) für die wichtigsten Objekte der Software:

Objekt	Operation	Beschreibung	Wert
KNX-Objekt	SETVALUE	Sendet einen entsprechenden Wert an die KNX Gruppenadresse	Hängt vom KNX-Objekt ab Beispiele: 0 → AUS 1 → EIN 50 → setze auf 50%
	GETVALUE	Sendet eine Anfrage, um den aktuellen Status zu erhalten	Kein Wert
Szenario	EXECUTE	Führt das Szenario aus	Kein Wert
	STOPEXECUTION	Unterbricht das Szenario (wenn in Ausführung)	Kein Wert
Logik	EVALUATE	Wertet die Logik aus und setzt deren Ausgänge neu	Kein Wert
Bedingung	CALCULATE	Wertet die Bedingung aus und setzt deren Ausgänge neu	Kein Wert
On-Screen-Nachricht	INSERT	Setzt die Nachricht in die Log-Tabelle, damit sie in der Nachrichtenzentrale angezeigt wird	Kein Wert
E-Mail-Nachricht	SENDMAIL	Versendet die E-Mail-Nachricht	Kein Wert
Integrator	INTEGRATE	Berechnet den Wert des Integrators neu	Kein Wert
	RESET	Setzt den Integrator auf 0 zurück	Kein Wert
Virtuelles Objekt	SETVALUE	Setzt den Wert des virtuellen Objekts	Hängt vom Typ des virtuellen Objekts ab

Um den Umgang mit den meistverwendeten Objekten (KNX-Objekte, Szenarien und virtuelle Objekte) zu erleichtern, wurden folgende "Abkürzungen", sprich Funktionen, die direkt auf der PHP-Variablen, die die Objekt-Referenz enthalten, ausgeführt werden können, eingebaut:

```
$object->set(...);  //Setzt den als Parameter übergebenen Wert (SEVALUE-Operation)
$object->run();     //Führt das Szenario aus
$object->stop();    //Unterbricht das Szenario
```

Soll z.B. das bereits mehrfach erwähnte Beispiel-Szenario gestartet werden, reichen folgende Aufrufe innerhalb des Scripts vollkommen aus:

```
$scenarioFilm = objM::objGet(548);
$scenarioFilm->run();
```

5.6 AKTUALISIERUNG VON OBJEKTEN IN DER DATENBANK

Folgende Funktion erlaubt die Versendung eines SQL-Befehls, welcher in der Datenbank-Tabelle der Objekte die Eigenschaft des angegebenen Objekts aktualisieren kann:

```
objM::objUpdateToDb($objectId,$field,$value);
```

Folgende sind die verfügbaren Eigenschaften:

$objectId	ID des zu aktualisierenden Objekts
$field	Name der Spalte der Datenbank, welche aktualisiert werden soll
$value	Neu zu setzender Wert

Auf diese Art und Weise kann z.B. der Name eines Objektes verändert werden…

```
objM::objUpdateToDb(XXX,"name","Neuer Name");
```

…wobei XXX natürlich für die ID des zu aktualisierenden Objekts steht. Natürlich kann dieselbe Funktion auch direkt über ein Objekt aufgerufen werden. In dem Fall lautet die korrekte Syntax:

```
$object->updateToDb($field,$value);
```

Ab Version 1.1 ist es zudem möglich, Attribute eines Objektes direkt während der Ausführung des Scripts aus der Datenbank auszulesen und deren Aktualisierung zu forcieren:

```
$object->refreshFromDb($field);
```

Um z.B. den Wert zu aktualisieren, kann folgender Befehl verwendet werden:

```
$object->refreshFromDb("value");
```

6 "SURROUNDING" - BIBLIOTHEK

6.1 EINLEITUNG

Die Bibliothek "surrounding" stellt dem Autor des Scripts diverse Objekte und Funktionen zur Verfügung, über welche die Interaktion mit dem Run-Script sowie den als EINGÄNGE bzw. AUSGÄNGE verknüpften Objekten gesteuert werden kann.

Diese Bibliothek hängt direkt von der zuvor gesehenen "object" – Bibliothek ab; die Struktur der Objekte sowie der Verknüpfungen in dieser Bibliothek lehnt sich somit an Kapitel 5 an, auf welches für manches Detail direkt verwiesen wird.

Wird die "**surrounding**" – Bibliothek in einem Script eingebunden, muss die "**object**" – Bibliothek NICHT separat eingebunden werden, da dies bereits automatisch geschieht.

6.2 EINBINDUNG DER BIBLIOTHEK

Um die Bibliothek in ein Script einzubinden, reicht folgender Aufruf zu Beginn des Scripts:

```
include_library("surrounding");
```

6.3 UMGEBUNG DES RUN-SCRIPTS

Von der nächsten Zeile an kann im Script folgendes Objekt verwendet werden:

```
$me
```

Dieses Objekt ist identisch strukturiert wie die in Kapitel 5 beschriebenen Objekte und ist eine direkte Referenz auf das Run-Script, welches das Script ausführt (und erlaubt somit eine Interaktion mit demselben). Das Objekt stellt natürlich auch die verknüpften Objekte im EINGANG ("PARENT-Objekte") und AUSGANG ("CHILD-Objekte") über Arrays zur Verfügung, welche im Eigenschaftsfenster des Run-Scripts definiert worden sind (siehe Kapitel 4.2)

```
$me->parentRelations[];    //Verknüpfungen EINGÄNGE
$me->childRelations[];     //Verknüpfungen AUSGÄNGE
```

Zur Erleichterung können die EINGÄNGE des Run-Scripts ("PARENT-Objekte") direkt über folgende Variablen angesprochen werden:

```
$in1;       //Erster Eingang des Run-Scripts
$in2;       //Zweiter Eingang des Run-Scripts
…
```

Auf dieselbe Art und Weise können auch die AUSGÄNGE des Run-Scripts ("CHILD-Objekte") direkt angesprochen werden:

```
$out1;       //Erster Ausgang des Run-Scripts
$out2;       //Zweiter Ausgang des Run-Scripts
…
```

Wie bereits in Kapitel 5.4 empfohlen, kann in der Debug-Phase des Scripts die gesamte Struktur des Run-Scripts über folgenden Aufruf angezeigt werden

```
echo "</pre>"; print_r($me); echo "<pre>";
```

6.4 BEISPIELE

Will man ein Script erstellen, welches als AUSGANGSWERT die Summe der Werte zweier EINGÄNGE angezeigt, kann das (zusätzlich zur Verknüpfung der entsprechenden Objekte im Eigenschaftsfenster des Run-Scripts) wie folgt realisiert werden:

```
include_library("surrounding");          //Einbindung Bibliothek
$sum = $in1->value + $in2->value;        //Summe der Werte der ersten beiden Eingänge
output($sum);                            //Rückgabe des berechneten Wertes
```

Soll der Wert des ersten AUSGANGS hingegen direkt über das Script gesetzt werden und nicht über den AUSGANGSWERT, welcher nach Beendigung des Scripts zurückgegeben wird, so muss das Script folgendermaßen umgewandelt werden

```
include_library("surrounding");          //Einbindung Bibliothek
$sum = $in1->value + $in2->value;        //Summe der Werte der ersten beiden Eingänge
$out1->set($sum);                        //Setzen des Wertes des ersten Ausgangs
output();                                //Beendigung des Scripts ohne Rückgabe
```

In diesem letzten Fall ist es wichtig, die Funktion output() OHNE Parameter aufzurufen, damit das Run-Script die verknüpften Ereignisse NICHT ausführt (da der Wert ja bereits innerhalb des Scripts gesetzt wurde). Ansonsten würde der Ausgang zweimal angesteuert werden: einmal vom Script selbst und einmal durch die Ausführung des verknüpften Ereignisses im Ausgang des Run-Scripts.

Soll hingegen ein Script erstellt werden, welches einfach die Werte sämtlicher im Eingang verknüpfter Objekte summiert, kann folgender Code zum Einsatz kommen:

```
include_library("surrounding");                    //Einbindung Bibliothek
$sum = 0;                                          //Initialisierung Summe auf 0
foreach($me->parentRelations as $parentRelation)   //Prüfung der Verknüpfungen im Eingang
{
    $parent = $parentRelation->parent;            //Identifizierung des Objekts im Eingang
    $sum = $sum + $parent->value;                 //Hinzufügen des Wertes zum vorigen Wert
}
output($sum);                                      //Rückgabe des berechneten Wertes
```

Auf dieselbe Art und Weise wurden die in Ihrem **KNX**CONTROL-Gerät vorhandenen Beispiel-Scripts AND und OR realisiert; weitere Informationen dazu finden Sie im Kapitel über die enthaltenen Beispiel-Scripts.

7 "SERIAL" - BIBLIOTHEK

7.1 EINLEITUNG

Die Bibliothek "serial" erlaubt das Lesen und Schreiben von Daten über die RS232-Schnittstelle Ihres **KNX**CONTROL-Gerätes oder, in Kombination mit einem geeigneten USB-RS232 Adapter, auch über einen der vorhandenen USB-Ports.

7.2 EINBINDUNG DER BIBLIOTHEK

Um die Bibliothek in ein Script einzubinden, reicht folgender Aufruf zu Beginn des Scripts:

```
include_library("serial");
```

Die Bibliothek stellt folgende Klasse zur Verfügung…

```
$serialM
```

… über welche Operationen auf der RS232-Schnittstelle durchgeführt[3] werden können, wie in den weiteren Kapiteln beschrieben.

[3] In Unterschied zu den bisher gesehenen Klassen, welche statisch aufgerufen werden, muss die Klasse $serialM wie ein Objekt aufgerufen werden, mit dem Trennzeichen "->" anstelle von "::" vor dem Funktionsaufruf.

7.3 INITIALISIERUNG DER SCHNITTSTELLE

Die Klasse **serialM** erlaubt die Konfiguration der primären Eigenschaften der seriellen Schnittstelle; folgende Tabelle zeigt die Eigenschaften, die Funktion zum Zugreifen, die Standard-Werte sowie die möglichen alternativen Werte:

Eigenschaft	Aufruf / Funktion	Standard	Mögliche Werte
COM-Port	`$serialM->deviceSet(...);`	"/dev/ttyS4" HINWEIS: RS232 von **KNX**SERVER	"/dev/ttyS4" (RS232) "/dev/dlabusb1" (USB1) "/dev/dlabusb2" (USB2)
Baudrate	`$serialM->confBaudRate(...);`	19200	110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
Parität	`$serialM->confParity(...);`	"none"	"none", "odd", "even"
Daten-Bits	`$serialM->confCharacterLength(...);`	8	8, 7
Stop-Bit	`$serialM->confStopBits();`	1	1, 1.5, 2
Flusskontrolle	`$serialM->confFlowControl();`	"none"	"none", "rts/cts", "xon/xoff"

Sollten die Standard-Eigenschaften nicht verändert werden müssen, können die folgenden Funktionen ignoriert werden. Ansonsten müssen die gewünschten Einstellungen über die folgenden Funktionen gesetzt werden, bevor die Schnittstelle für Lese-/Schreibvorgänge geöffnet wird.

Beispiele:

```
include_library("serial");
$serialM->confBaudRate(9600);
$serialM->confParity("odd");
$serialM->confFlowControl("rts/cts");
...
```

7.4 SCHREIBVORGÄNGE

Um Schreibvorgänge starten zu können, muss die Schnittstelle zuerst über folgenden Befehl geöffnet werden:

```
$serialM->deviceOpen();
```

Im Anschluss kann über den Befehl...

```
$serialM->sendMessage(...);
```

... eine Zeichenkette übergeben werden, welche an die Schnittstelle gesendet wird. Ev. kann dem Befehl auch ein End-Zeichen übergeben werden, wenn vom empfangenden Gerät benötigt.

Beispiele:

```
$serialM->sendMessage("test");          //Schreiben von "test" ohne End-Zeichen
$serialM->sendMessage("test\n");        //Schreiben von "test" und Zeilensprung
$serialM->sendMessage("test".chr(13)); //Schreiben von "test" und ENTER
```

Der Aufruf sieht auch einen 2. Parameter vor, über welchen eine Wartezeit (in Sekunden) angegeben werden kann, welche vor dem Auslesen einer Antwort verstreichen muss (Details zum Auslesen von Daten der seriellen Schnittstelle sind im nächsten Kapitel zu finden):

```
$serialM->sendMessage("test\n",2);      //Schreiben und Warten von 2 Sekunden
$serialM->sendMessage("test\n",0.1);    //Schreiben und Warten von 100 Millisekunden
```

Nach Abschluss des Schreibvorganges muss die Schnittstelle über folgenden Befehl wieder geschlossen werden:

```
$serialM->deviceClose();
```

7.5 LESEVORGÄNGE

Auch vor Lesevorgängen muss die Schnittstelle zuerst über folgenden Befehl geöffnet werden:

```
$serialM->deviceOpen();
```

Das Auslesen von Daten kann z.B. nach einem Schreibvorgang mit Wartezeit erfolgen, wie zuvor bereits erwähnt. Um Daten auszulesen, muss eine der folgenden Methoden verwendet werden:

```
$value = $serialM->readPort();        //Liest solange Daten von der RS232, wie verfügbar
$value = $serialM->readPort(512);     //Liest eine bestimmte Anzahl von Bytes
```

Nach Abschluss des Lesevorganges muss die Schnittstelle wieder geschlossen werden, wie bereits zuvor gesehen:

```
$serialM->deviceClose();
```

7.6 DIREKTER ZUGRIFF AUF DIE SCHNITTSTELLE

Alternativ zu den Befehlen der "serial" Bibliothek können Schreibvorgänge auf die serielle Schnittstelle auch über native PHP-Befehle erfolgen:

```
$fp = fopen('/dev/ttyS4','r+b');    //Öffnet den RS232-Port im Schreibmodus
$msg = "...";                       //Initialisierung der Nachricht
fwrite($fp,$msg . Chr(13));         //Versand der Nachricht und ENTER
fclose($fp);                        //Schließt den Port erneut
```

Werden diese Befehle nach der Initialisierung der seriellen Schnittstelle (siehe Kapitel 7.3) ausgeführt, kann garantiert werden, dass die korrekten Port-Einstellungen zur Anwendung kommen.

8 "MODBUS" - BIBLIOTHEK

8.1 EINLEITUNG

Die Bibliothek "modbus" erlaubt die Verwaltung eines oder mehrerer MODBUS SLAVE – Geräte innerhalb eines Scripts, über das Netzwerk (Protokoll MODBUS TCP oder UDP).

8.2 EINBINDUNG DER BIBLIOTHEK

Um die Bibliothek in ein Script einzubinden, reicht folgender Aufruf zu Beginn des Scripts:

```
include_library("modbus");
```

Von der nächsten Zeile an kann im Script folgende Klasse verwendet werden:

```
modbusM
```

Diese Klasse ("MODBUS Manager" bzw. "MODBUS-Verwalter") stellt diverse Funktionen (bzw. "Methoden") zur Verfügung, um Objekte der Software zu laden und Operationen auszuführen. Die Klasse **modbusM** wird, wie bereits bei anderen Klassen gesehen, über "statische Aufrufe" verwendet, wie z.B.:

```
modbusM::bind(...);
```

Wie gezeigt, erfolgen die statischen Aufrufe beginnend mit dem Namen der Klasse (ohne das typische Variablen-Präfix "$"), gefolgt von zwei Doppelpunkten und dem Namen der aufzurufenden Funktion; diese Aufrufe unterscheiden sich von normalen Objekt-Aufrufen (deren Name immer das Präfix "$" aufweist), welche das Trennzeichen "->" vor dem Namen der aufzurufenden Funktion benötigen.

8.3 ZUWEISUNG DES "MODBUS SLAVE" - GERÄTES

Bevor Lese-/Schreibzugriffe erfolgen können, muss der Klasse **modbusM** das MODBUS SLAVE –Gerät zugewiesen werden, mit welchem sie kommunizieren soll. Dies erfolgt über den Befehl

```
modbusM::bind($host,$protocol);
```

wobei

$host	IP-Adresse des Gerätes
$protocol	zu verwendendes Protokoll, entweder "TCP" (Standard) oder "UDP"

Solange dieser Befehl nicht erneut aufgerufen wird, werden jegliche folgende Operationen an das bisher gewählte MODBUS SLAVE – Gerät gesendet.

8.4 AUSLESEN DER REGISTER

Um ein oder mehrere Register vom MODBUS SLAVE – Gerät auszulesen, muss folgender Befehl verwendet werden

```
$data = modbusM::readRegisters($unitID, $reference, $quantity);
```

wobei

$unitID	ID des auszulesenden Gerätes. Wenn nicht vom Hersteller anders verlangt, 0 (Null) verwenden
$reference	Auszulesendes Register. HINWEIS: die zu verwendende Nummer muss das Gerät-interne Offset berücksichtigen, in Abhängigkeit der internen Nummerierung. Normalerweise liest "0" z.B. das interne Register 40001 aus.
$quantity	Anzahl der auszulesenden Register. 1 angeben, wenn immer nur ein Register ausgelesen werden soll.

Das Resultat wird von der Funktion in Form eines Byte-Arrays (oder, in anderen Worten, einer Zeichenkette) zurückgegeben; weitere Informationen zur Konvertierung der erhaltenen Daten finden Sie in Kapitel 8.9 dieses Handbuchs.

8.5 AUSLESEN DER COILS

Um ein oder mehrere Coils (Binär-Ein- bzw.- Ausgänge) vom MODBUS SLAVE – Gerät auszulesen, muss folgende Funktion verwendet werden:

```
$data = modbusM::readCoils($unitID, $reference, $quantity);
```

wobei

$unitID	ID des auszulesenden Gerätes. Wenn nicht vom Hersteller anders verlangt, 0 (Null) verwenden
$reference	Auszulesender Coil. HINWEIS: die zu verwendende Nummer muss das Gerät-interne Offset berücksichtigen, in Abhängigkeit der internen Nummerierung.
$quantity	Anzahl der auszulesenden Coils. 1 angeben, wenn immer nur ein Coil ausgelesen werden soll.

Das Resultat wird von der Funktion in Form eines Arrays mit Bool-Werten (ein Wert für jeden Coil) zurückgegeben.

8.6 SCHREIBEN VON REGISTERN

Um ein oder mehrere Register des MODBUS SLAVE – Gerätes zu schreiben, muss folgender Befehl verwendet werden

```
$data = modbusM::writeRegisters($unitID, $reference, $data, $dataTypes);
```

wobei

$unitID	ID des zu verbindenden Gerätes. Wenn nicht vom Hersteller anders verlangt, 0 (Null) verwenden
$reference	Zu schreibendes Register. HINWEIS: die zu verwendende Nummer muss das Gerät-interne Offset berücksichtigen, in Abhängigkeit der internen Nummerierung
$data	Werte-Array, welches den Registern übergeben werden soll (ein Wert für jedes zu schreibende Register). Die Anzahl der Werte bestimmt die Anzahl der zu schreibenden Register.
$dataTypes	Format-Array für jedes zu schreibende Register (passend zu den Elementen in **$data**). Mögliche Werte: • "WORD" • "INT" • "DINT" • "REAL"

Die Funktion gibt einen Return-Code zurück: dieser ist gleich _SCRIPT_RESULT_NOERROR (0) bei erfolgreicher Durchführung oder aber ein Fehlercode. Weitere Informationen zu den möglichen Return-Codes finden Sie in Kapitel 2.5 dieses Handbuchs.

Folgender Code zeigt ein Beispiel für die eben vorgestellte Funktion:

```
$data = array(10,-1000,2000,3.0);
$dataTypes = array("WORD","INT","DINT","REAL");
$result = modbusM::writeRegisters(0, 12288, $data, $dataTypes);
```

8.7 SCHREIBEN VON COILS

Um ein oder mehrere Coils des MODBUS SLAVE – Gerätes zu schreiben, muss folgender Befehl verwendet werden

```
$data = modbusM::writeCoils($unitID, $reference, $data);
```

wobei

$unitID	ID des zu verbindenden Gerätes. Wenn nicht vom Hersteller anders verlangt, 0 (Null) verwenden
$reference	Zu schreibender Coil. HINWEIS: die zu verwendende Nummer muss das Gerät-interne Offset berücksichtigen, in Abhängigkeit der internen Nummerierung
$data	Array mit Bool-Werten, die den Coils zugewiesen werden sollen. Die Anzahl der Werte bestimmt die Anzahl der zu schreibenden Coils.

Die Funktion gibt einen Return-Code zurück: dieser ist gleich _SCRIPT_RESULT_NOERROR (0) bei erfolgreicher Durchführung oder aber ein Fehlercode. Weitere Informationen zu den möglichen Return-Codes finden Sie in Kapitel 2.5 dieses Handbuchs.

Folgender Code zeigt ein Beispiel für die eben vorgestellte Funktion:

```
$data = array(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE, TRUE, TRUE,
TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, FALSE, FALSE,
            FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE,
            TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE);
$result = modbusM::writeCoils(0, 12288, $data);
```

8.8 LESEN UND SCHREIBEN VON REGISTERN

Um ein oder mehrere Register des MODBUS SLAVE – Gerätes zu schreiben und gleichzeitig ein oder mehrere Register mit derselben Verbindung auslesen zu können, muss folgender Befehl verwendet werden

```
$data = modbusM::readWriteRegisters($unitID, $referenceRead, $quantity, $referenceWrite,
$data, $dataTypes);
```

wobei

$unitID	ID des zu verbindenden Gerätes. Wenn nicht vom Hersteller anders verlangt, 0 (Null) verwenden
$referenceRead	Auszulesendes Register
$quantity	Anzahl der auszulesenden Register
$referenceWrite	Zu schreibendes Register
$data	Werte-Array, welches den Registern übergeben werden soll (ein Wert für jedes zu schreibende Register). Die Anzahl der Werte bestimmt die Anzahl der zu schreibenden Register.
$dataTypes	Format-Array für jedes zu schreibende Register (passend zu den Elementen in **$data**). Mögliche Werte: • "WORD" • "INT" • "DINT" • "REAL"

Die Funktion gibt die gelesenen Daten in derselben Form zurück wie in Kapitel 8.4 dieses Handbuchs angegeben; die Bedeutung der zu übergebenden Parametern entspricht exakt jener der Parameter aus Kapitel 8.4 und 8.6.

8.9 UMWANDLUNG DER EINGELESENEN WERTE

Die Werte, die von den Lesefunktionen der Register erhalten werden, können zur weiteren Verwendung ins PHP-Format konvertiert werden, in Abhängigkeit des Originalformats sowie des jeweiligen Inhalts (siehe Dokumentation des verwendeten MODBUS Gerätes). Es folgen einige Beispiele, in welchen **$data** immer das ausgelesene Byte-Array ist (dessen Länge von der Anzahl der auszulesenden Registern abhängt, wie im entsprechenden Parameter der Lese-Funktion definiert).

8.9.1 4 BYTE - WERTE

Sind die Werte innerhalb der Register des Gerätes im 4 Byte-Format angelegt (2 Register für jeden Wert), können diese über folgende Funktion in ein Array zerlegt werden:

```
$values = array_chunk($data, 4);
```

Nun kann auf jeden Wert des Arrays eine der folgenden Konvertierungs-Funktionen angewendet werden, in Abhängigkeit vom Original-Format des MODBUS – Gerätes. Es folgen einige Beispiele:

Konvertierung der Werte von REAL in FLOAT:

```
foreach($values as $bytes)
{
   $value = PhpType::bytes2float($bytes);
[...]
}
```

Konvertierung der Werte von DINT in INTEGER:

```
foreach($values as $bytes)
{
    $value = PhpType::bytes2signedInt($bytes);
   [...]
}
```

Konvertierung der Werte von DWORD in INTEGER:

```
foreach($values as $bytes)
{
    $value = PhpType::bytes2unsignedInt($bytes);
   [...]
}
```

8.9.2 2 BYTE - WERTE

Sind die Werte innerhalb der Register des Gerätes im 2 Byte-Format angelegt (1 Register für jeden Wert), können diese über folgende Funktion in ein Array zerlegt werden:

```
$values = array_chunk($data, 2);
```

Nun kann auf jeden Wert des Arrays eine der folgenden Konvertierungs-Funktionen angewendet werden, in Abhängigkeit vom Original-Format des MODBUS – Gerätes. Es folgen einige Beispiele:

Konvertierung der Werte von INT in INTEGER

```
foreach($values as $bytes)
{
   $value = PhpType::bytes2signedInt($bytes);
[...]
}
```

Konvertierung der Werte von WORD in INTEGER

```
foreach($values as $bytes)
{
    $value = PhpType::bytes2unsignedInt($bytes);
   [...]
}
```

Soll hingegen eine korrekte Formatierung der gesamten ausgelesenen Register im String-Format (Zeichenkette) erfolgen, kann folgender Befehl verwendet werden:

```
PhpType::bytes2string($data);
```

9 "SONOS" – BIBLIOTHEK

9.1 EINLEITUNG

Die Bibliothek "sonos" erlaubt die Verwaltung eines oder mehrerer SONOS – Geräte (Multiroom) innerhalb eines Scripts. Details zum SONOS-System finden Sie auf der Homepage des Herstellers: www.sonos.com.

Diese Bibliothek ist ab Version 1.1 der **KNX**CONTROL Produkte verfügbar. Sollten Sie diese Bibliothek nicht vorfinden, aktualisieren Sie Ihr **KNX**CONTROL-Gerät bitte auf die letzte Version.

9.2 EINBINDUNG DER BIBLIOTHEK

Um die Bibliothek in ein Script einzubinden, reicht folgender Aufruf zu Beginn des Scripts:

```
include_library("sonos");
```

Von der nächsten Zeile an kann im Script folgende Klasse verwendet werden:

```
sonosM
```

Diese Klasse ("SONOS Manager" bzw. "SONOS-Verwalter") stellt diverse Funktionen (bzw. "Methoden") zur Verfügung, um Objekte der Software zu laden und Operationen auszuführen. Die Klasse **sonosM** wird, wie bereits bei anderen Klassen gesehen, über "statische Aufrufe" verwendet, wie z.B.:

```
sonosM::bind(...);
```

Wie gezeigt, erfolgen die statischen Aufrufe beginnend mit dem Namen der Klasse (ohne das typische Variablen-Präfix "$"), gefolgt von zwei Doppelpunkten und dem Namen der aufzurufenden Funktion; diese Aufrufe unterscheiden sich von normalen Objekt-Aufrufen (deren Name immer das Präfix "$" aufweist), welche das Trennzeichen "->" vor dem Namen der aufzurufenden Funktion benötigen.

9.3 ZUWEISUNG DES "SONOS"-GERÄTS

Bevor Lese-/Schreibzugriffe erfolgen können, muss der Klasse **sonosM** das SONOS–Gerät zugewiesen werden, mit welchem sie kommunizieren soll. Dies erfolgt über den Befehl

```
sonosM::bind($host);
```

wobei

$host IP-Adresse des Gerätes

Solange dieser Befehl nicht erneut aufgerufen wird, werden jegliche folgende Operationen an das bisher gewählte SONOS – Gerät gesendet.

Um die IP-Adresse eines SONOS-Gerätes zu erfahren, muss auf dem eigenen PC/MAC die Kontrollsoftware von SONOS installiert werden. Ruft man das Informationsfenster der Anlage auf, wird die IP-Adresse sämtlicher konfigurierter SONOS-Geräte angezeigt:

9.4 BEDIENUNG DES "SONOS"-GERÄTS

Nachdem das gewünschte SONOS-Gerät über den "bind"-Befehl verknüpft worden ist, kann es über folgende Befehle angesteuert werden:

```
sonosM::stop();          //Stoppt die Wiedergabe der Multimedia-Inhalte
sonosM::play();          //Startet die Wiedergabe der Multimedia-Inhalte
sonosM::pause();         //Pausiert die Wiedergabe der Multimedia-Inhalte
```

Die Auswahl der Multimedia-Inhalte, welche für die Wiedergabe verwendet werden sollen (Playlist bzw. Bibliothek), muss über die Kontrollsoftware von SONOS eingestellt werden.

Innerhalb einer Playlist kann über folgende Befehle eine Umschaltung der Tracks erfolgen:

```
sonosM::next();           //Track weiter
sonosM::previous();        //Track zurück
sonosM::setTrack($number);  //Springe zum Track mit Nummer $number
```

Es ist zudem möglich, einen Track um eine bestimmte Zeit (in Sekunden) vorzuspringen, und zwar über folgenden Befehl:

```
sonosM::setTime($time);
```

Der Befehl, um wieder zum Anfang des Tracks zurückzuspringen, lautet:

```
sonosM::rewind();
```

Dies hat dieselbe Wirkung wie der Befehl `sonosM::setTime(0);` .

Zuletzt ist es auch möglich, den aktuellen Status des SONOS-Gerätes über folgenden Befehl abzufragen:

```
$state = sonosM::getState();
```

Der Rückgabewert dieser Funktion ist einer der folgenden:

- 1 → Wiedergabe aktiv (PLAY)
- 2 → Wiedergabe pausiert (PAUSE)
- 3 → Wiedergabe gestoppt (STOP)

9.5 LAUTSTÄRKE-REGELUNG

Die Lautstärke des SONOS-Gerätes kann über folgenden Befehl angesteuert werden…

```
sonosM::setVolume($volume);
```

…wobei *$volume* eine Ganzzahl zwischen 0 und 100 annehmen kann; des Weiteren kann auch die aktuelle Lautstärke über folgenden Befehl abgefragt werden:

```
$volume = sonosM::getVolume();
```

Auch hier ist der Rückgabewert eine Ganzzahl zwischen 0 und 100.

Die Lautstärke kann temporär auch über den MUTE-Befehl (Stummschaltung) auf 0 gesetzt werden; der MUTE-Status kann zudem auch jederzeit ausgelesen werden:

```
sonosM::setMute();               //Aktiviert die "MUTE"-Funktion
$mute = sonosM::getMute();       //Liest den aktuellen "MUTE"-Status aus
```

9.6 WIEDERGABE-MODUS

Über folgenden Befehl kann der Wiedergabe-Modus der aktuellen Playlist verändert werden:

```
sonosM::setPlayMode($mode);       //Wiedergabe-Modus einstellen
```

Als Argument *$mode* muss einer der folgenden Werte übergeben werden:

- 0 → Normal (die Wiedergabe stoppt nach dem letzten Track)
- 1 → Wiederholen (nach dem letzten Track beginnt die Playlist wieder von vorne)
- 2 → Zufall (die Tracks werden kontinuierlich im Zufallsmodus abgespielt)

9.7 MULTIMEDIA-INFOS

Es ist jederzeit möglich, Informationen zum Multimedia-Inhalt abzurufen, welcher gerade abgespielt wird. Der Befehl hierfür lautet:

```
$info = sonosM::getMediaInfo();       //Multimedia-Informationen abrufen
```

Die Rückgabe erfolgt in Form eines Arrays, welches u.a. folgende Informationen enthält:

artist	Künstler
title	Titel des Tracks
album	Name des Albums
Track	Tracknummer
position	Aktuelle Wiedergabezeit des Tracks (in Sekunden)
duration	Gesamte Dauer des Tracks (in Sekunden)

Um eine spezielle Information aus dem erhaltenen Array weiterverwenden zu können, gehen Sie wie folgt vor:

```
$info = sonosM::getMediaInfo();       //Ruft das Info-Array ab
$title = $info["title"];             //Weist den Titel der Variablen $title zu
$album = $info["album"];             //Weist den Albumnamen der Variablen $album zu
...
```

9.8 KONFIGURATIONSBEISPIEL

Ihr **KNX**CONTROL-Gerät bietet, zusammen mit der Einführung der SONOS-Bibliothek, eine Reihe von Beispielen, welche die Verwendung der wichtigsten Funktionen dieses Kapitels in praktischer Anwendung zeigen.

Diese Beispiele, welche in Kapitel 11 genauer erklärt werden, können dazu verwendet werden, in Kombination mit einem KOMPLEXEN OBJEKT einen SONOS-Baustein für die VISUALISIERUNG zu erstellen. Die nächsten Seiten erklären die dazu benötigten Schritte!

9.8.1 ERSTELLUNG KOMPLEXES OBJEKT

Als erster Schritt muss ein komplexes Objekt für jedes SONOS-Gerät erstellt werden, welches über die VISUALISIERUNG gesteuert werden soll. Als Template muss „MULTIROOM – ZONE“ eingestellt werden; Informationen zur Handhabung von komplexen Objekten finden Sie im INSTALLATIONSHANDBUCH Ihres **KNX**CONTROL-Gerätes.

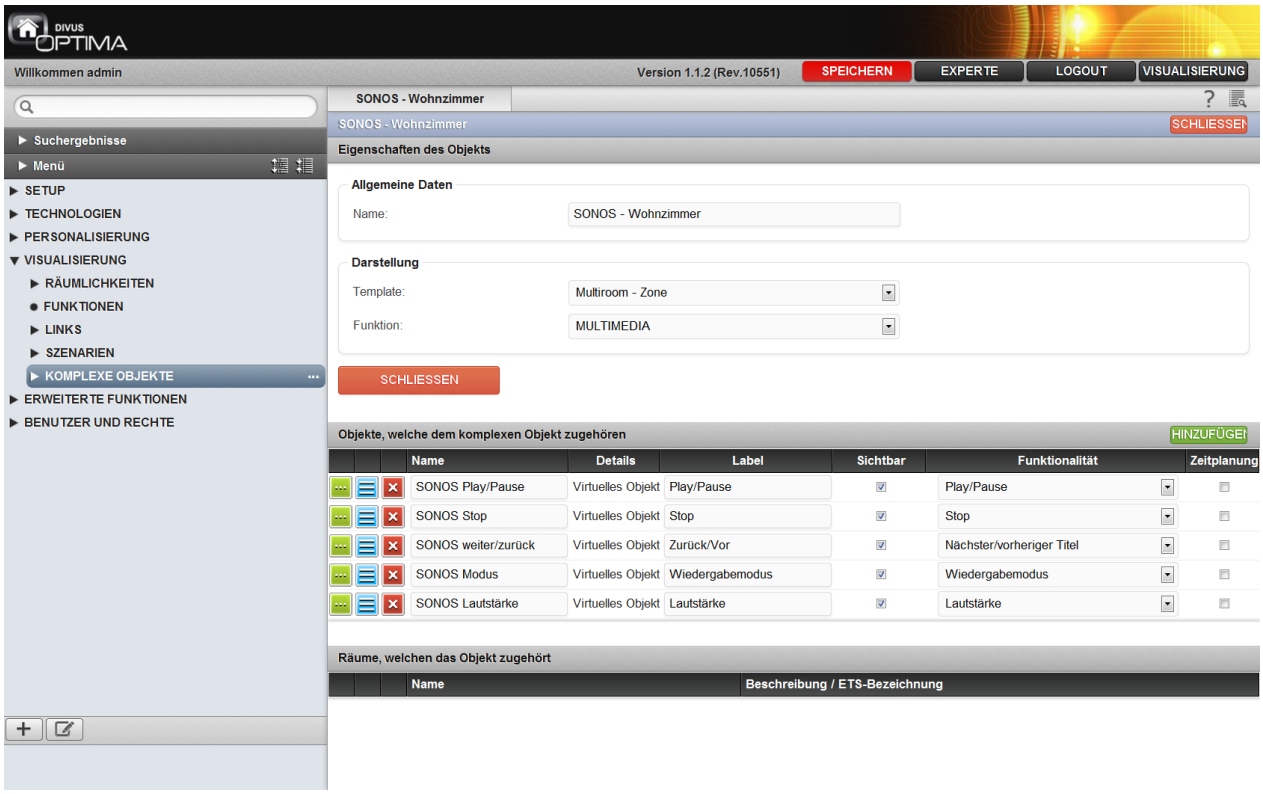
Nach Erstellung des komplexen Objekts und Einstellung des Templates, muss wiederholt auf den Button „HINZUFÜGEN“ im Bereich „OBJEKTE, WELCHE DEM KOMPLEXEN OBJEKT ZUGEHÖREN“ geklickt werden, um dem komplexen Objekt VIRTUELLE OBJEKTE zuzuweisen. Die Anzahl der virtuellen Objekte hängt von der Anzahl der Funktionen ab, welche man steuern möchte. Das komplexe Objekt stellt folgende Funktionen zur Verfügung (FUNKTIONALITÄT):

- Wiedergabe starten / pausieren(PLAY / PAUSE)
- Wiedergabe pausieren (STOP)
- Nächster / vorheriger Titel
- Wiedergabemodus
- Lautstärke

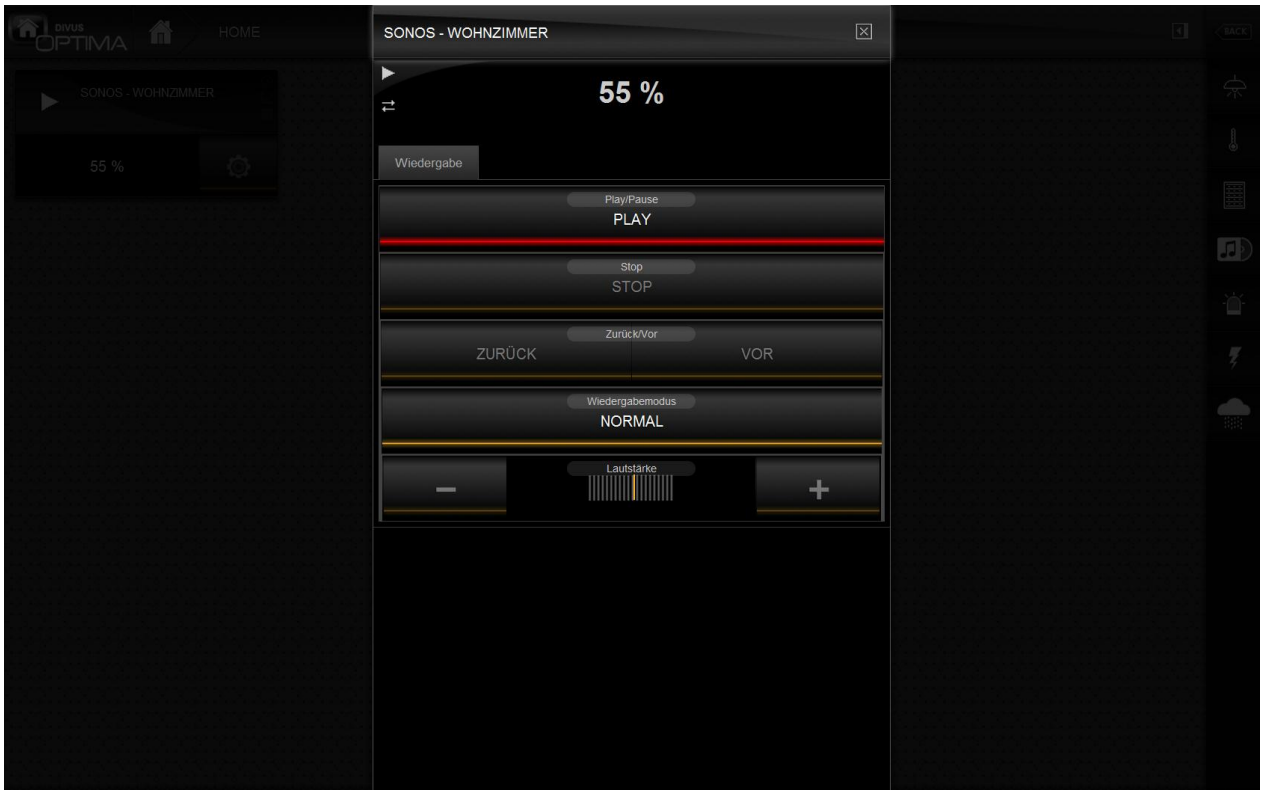
Um weitere Funktionen des SONOS-Systems steuern zu können, muss anstelle des “MULTIROOM – ZONE” – Objekts ein komplexes Objekt mit dem Template “ALLGEMEIN“ erstellt werden. Dessen Konfiguration wird im INSTALLATIONSHANDBUCH Ihres **KNX**CONTROL-Gerätes beschrieben.

Nach Erstellung der einzelnen Unter-Objekte muss diesen über die Spalte FUNKTIONALITÄT der Anwendungszweck und über die Spalte NAME eine Bezeichnung für das jeweilige Objekt zugewiesen werden. Zudem kann über die Spalte LABEL eine weitere Bezeichnung angegeben werden, welche im Anschluss im komplexen Objekt in der VISUALISIERUNG dargestellt wird.

Folgender Screenshot stellt das eben Beschriebene grafisch dar:



Wird das komplexe Objekt einer Räumlichkeit zugwiesen, kann es über die VISUALISIERUNG aufgerufen werden und präsentiert sich in folgender Art und Weise:



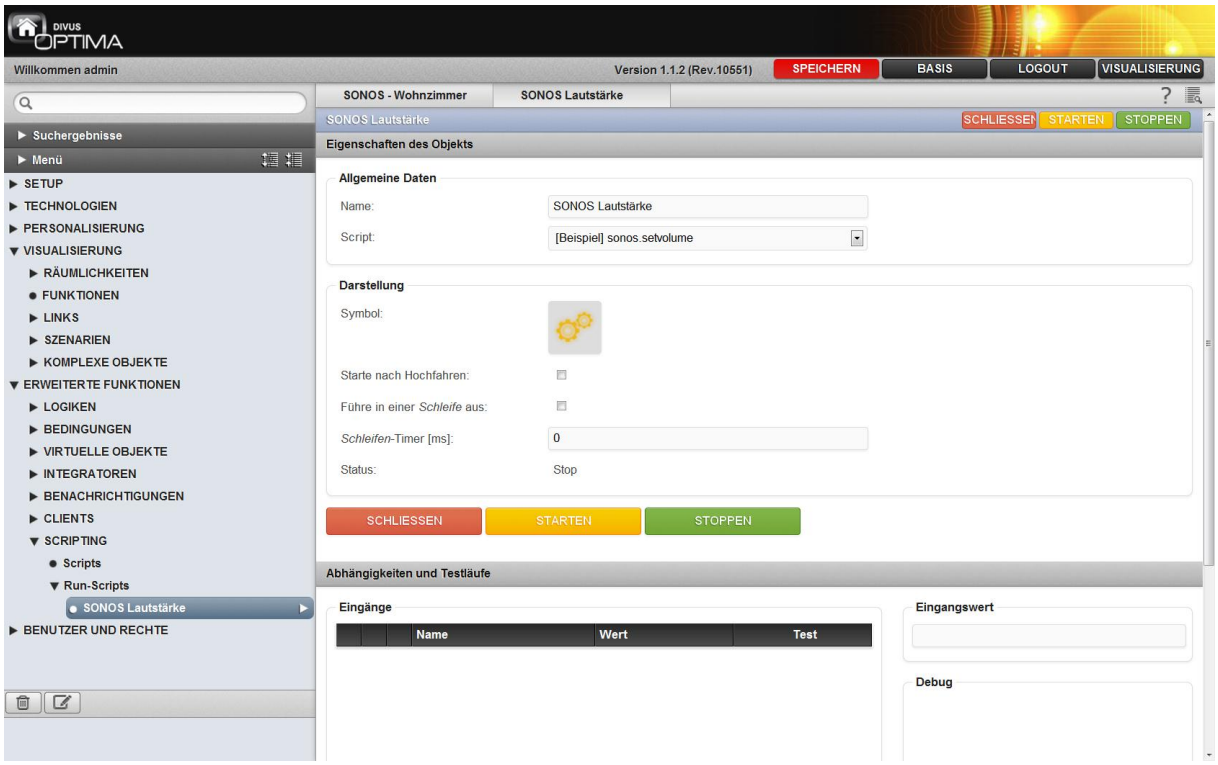
9.8.2 ERSTELLUNG UND VERKNÜPFUNG DER RUN-SCRIPTS

Nach Erstellung des komplexen Objekts hat man zwar ein aus grafischer Sicht vollständiges Objekt in der VISUALISIERUNG, das jedoch noch mit keinerlei Funktion belegt ist. Um die einzelnen Elemente nun mit der SONOS-Anlage zu verknüpfen, muss für jede Funktion des komplexen Objekts ein Run-Script erstellt werden, welches:

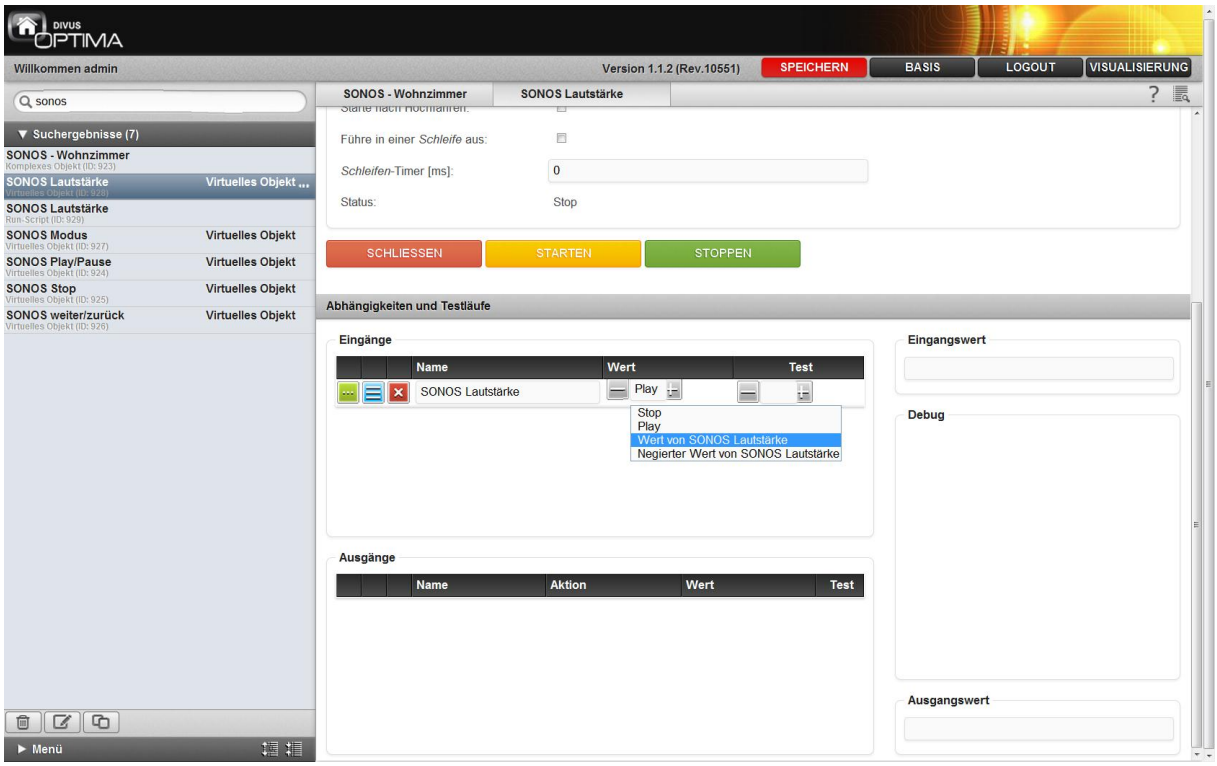
- Seinerseits mit einem Script verknüpft ist, welches die Funktionen der SONOS-Bibliothek nutzt
- Mit dem entsprechenden VIRTUELLEN OBJEKT des komplexen Objekts verknüpft wird. Das virtuelle Objekt muss dabei als EINGANG fungieren, damit die Befehle aus der VISUALISIERUNG an das Script weitergegeben werden und über dieses zur SONOS-Anlage gelangen.

Unter den in Ihrem **KNX**CONTROl-Gerät vorhandenen Beispielen finden sich bereits alle Scripts, welche zur Ansteuerung von SONOS-Geräten über das komplexe Objekt mit Template „MULTIROOM – ZONE" benötigt werden. Weitere Informationen zu diesen Scripts finden Sie in Kapitel 11 dieses Handbuches.

Als Beispiel für die Verknüpfung von virtuellem Objekt und Run-Script soll die Lautstärke-Regelung des SONOS-Gerätes dienen. Zu Beginn muss ein neues Run-Script erstellt werden, in welchem als SCRIPT das Beispiel "SONOS.SETVOLUME" über das Dropdown-Menü ausgewählt wird. Das Script muss weder nach dem Hochfahren noch in einer Schleife gestartet werden, weshalb die entsprechenden Checkboxen unverändert bleiben können.



Nun kann über die Suchfunktion eben jenes VIRTUELLE OBJEKT lokalisiert werden, welches zuvor im komplexen Objekt für die Lautstärke-Regelung erstellt worden ist (aus diesem Grund ist es wichtig, dass zuvor ein eindeutiger NAME für das Objekt vergeben worden ist), und via Drag&Drop in den Bereich EINGÄNGE des Run-Scripts gezogen werden. Als WERT muss der Wert des virtuellen Objekts selbst weitergegeben werden, was über das Dropdown-Menü entsprechend eingestellt werden kann:



Somit wird nun der Wert, der vom Benutzer in der VISUALISIERUNG als Lautstärke definiert wird, dem Run-Script als Eingangswert übergeben; dieses leitet den Wert an das verknüpfte Script weiter, in welchem der Wert über die entsprechende Funktion (in diesem Fall **setVolume()**) an das SONOS-Gerät übergeben wird.

10 "DUNE" – BIBLIOTHEK

10.1 EINLEITUNG

Die Bibliothek "dune" erlaubt die Verwaltung eines oder mehrerer DUNE-HD – Geräte (Multimedia/Multiroom) innerhalb eines Scripts. Details zum DUNE-System finden Sie auf der Homepage des Herstellers: http://dune-hd.com/eng/.

Diese Bibliothek ist ab Version 1.1 der **KNX**CONTROL Produkte verfügbar. Sollten Sie diese Bibliothek nicht vorfinden, aktualisieren Sie Ihr **KNX**CONTROL-Gerät bitte auf die letzte Version.

10.2 EINBINDUNG DER BIBLIOTHEK

Um die Bibliothek in ein Script einzubinden, reicht folgender Aufruf zu Beginn des Scripts:

```
include_library("dune");
```

Von der nächsten Zeile an kann im Script folgende Klasse verwendet werden:

```
duneM
```

Diese Klasse ("DUNE Manager" bzw. "DUNE-Verwalter") stellt diverse Funktionen (bzw. "Methoden") zur Verfügung, um Objekte der Software zu laden und Operationen auszuführen. Die Klasse **duneM** wird, wie bereits bei anderen Klassen gesehen, über "statische Aufrufe" verwendet, wie z.B.:

```
duneM::bind(...);
```

Wie gezeigt, erfolgen die statischen Aufrufe beginnend mit dem Namen der Klasse (ohne das typische Variablen-Präfix "$"), gefolgt von zwei Doppelpunkten und dem Namen der aufzurufenden Funktion; diese Aufrufe unterscheiden sich von normalen Objekt-Aufrufen (deren Name immer das Präfix "$" aufweist), welche das Trennzeichen "->" vor dem Namen der aufzurufenden Funktion benötigen.

10.3 ZUWEISUNG DES "DUNE"-GERÄTS

Bevor Lese-/Schreibzugriffe erfolgen können, muss der Klasse **duneM** das DUNE-HD–Gerät zugewiesen werden, mit welchem sie kommunizieren soll. Dies erfolgt über den Befehl

```
duneM::bind($host);
```

wobei

$host	IP-Adresse des Gerätes

Solange dieser Befehl nicht erneut aufgerufen wird, werden jegliche folgende Operationen an das bisher gewählte DUNE – Gerät gesendet.

10.4 ALLGEMEINE BEFEHLE ZUM "DUNE"-GERÄT

Die Kommunikation mit dem DUNE-Gerät erfolgt über Netzwerktelegramme, welche den Richtlinien des folgenden Protokolls entsprechen:

http://dune-hd.com/support/ip_control/dune_ip_control_overview.txt

Die genaue Spezifikation hängt zudem von der Firmware des eingesetzten DUNE-Gerätes ab.

Die „dune"-Bibliothek stellt folgenden, allgemeinen Befehl zur Verfügung:

```
$result = duneM::call($cmdString);
```

Über das Argument *$cmdString* kann der gewünschte Befehl in der Form übergeben werden, wie in der Dokumentation beschrieben; es muss lediglich der Part nach dem Zeichen "?" verwendet werden. Beispiel: steht in der Dokumentation…

```
http://10.0.0.1/cgi-bin/do?
cmd=start_file_playback&media_url=nfs://10.0.0.1:/VideoStorage:/SomeFolder/file.mkv
```

… so muss die Bibliothek folgenden Aufruf starten, um den Befehl umzusetzen:

```
duneM::call("cmd=start_file_playback&media_url=nfs://10.0.0.1:/VideoStorage:/SomeFolder/
file.mkv");
```

Erfolgt die Ausführung des Befehls korrekt, gibt die Funktion den Wert _SCRIPT_RESULT_NOERROR zurück, bei Fehlern ist der Rückgabewert hingegen _SCRIPT_RESULT_USERERROR.

Der Rückgabewert des zuletzt ausgeführten Befehls wird – wie im Dokument mit den Richtlinien des Protokolls beschrieben, in folgender globalen Variabel gespeichert:

```
$_duneLastResponse
```

Bei dieser Variablen handelt es sich um ein Objekt, welches sämtliche vom Gerät erhaltenen Informationen über seine Attribute zur Verfügung stellt. Das Resultat der durchgeführten Aktion ("ok" wenn fehlerlos durchgeführt, eine Fehlermeldung wenn nicht) kann über folgende Zeile im Script abgefragt werden:

```
$_duneLastResponse->command_status
```

10.5 BEDIENUNG DES "DUNE"-GERÄTS

Um die Handhabung der Scripts zur Steuerung von DUNE-Geräten zu vereinfachen, stellt die Bibliothek eine Reihe von Funktionen zur Verfügung, welche die wichtigsten Multimedia-Bedienoptionen bereits beinhalten und übersichtlicher zu verwenden sind.

Um z.B. einen bestimmten Multimedia-Inhalt wiederzugeben, kann – je nach Art des Inhalts – einer der folgenden Befehle verwendet werden:

```
duneM::playFile($url);           //Wiedergabe einer allgemeinen Film-Datei
duneM::playDVD($url);            //Wiedergabe einer DVD oder eines DVD-Abbildes
duneM::playBluRay($url);         //Wiedergabe einer BluRay oder eines BluRay-Abbildes
duneM::playList($url,$track);    //Wiedergabe einer Playlist, mit optionaler Angabe
                                   des Tracks, der zu Beginn abgespielt werden soll
```

Das Argument *$url* muss den Pfad zum Quell-Laufwerk (CD, DVD, BLU-RAY) oder zur Datei / der Playlist beinhalten, welche zur Wiedergabe genutzt werden soll. Dabei muss die Pfadangabe im vom DUNE-Protokoll geforderten Format erfolgen. Einige Beispiele:

```
duneM::playFile("nfs://10.0.0.1:/VideoStorage:/SomeFolder/file.mkv");
duneM::playDVD("smb://10.0.0.1/VideoStorage/SomeFolder/DVDFolder");
duneM::playDVD("storage_name://MyHDD1/SomeFolder/dvd_image.iso");
duneM::playBluRay("nfs://10.0.0.1:/VideoStorage:/SomeFolder/BlurayFolder");
duneM::playBluRay("nfs://10.0.0.1:/VideoStorage:/SomeFolder/bluray_image.iso");
```

Das DUNE-Gerät kann über folgende Befehle jederzeit auf einen "schwarzen Bildschirm" (zur Unterbrechung der Wiedergabe) oder das Hauptmenü geschalten werden:

```
duneM::blackScreen();
duneM::mainScreen();
```

Um das Gerät hingegen auszuschalten, kommt folgender Befehl zum Einsatz:

```
duneM::standby();
```

10.6 EMULATION FERNBEDIENUNG

Über die Bibliothek ist es ebenfalls möglich, die Befehle der DUNE-Fernbedienung zu emulieren. Dazu kann folgender Befehl verwendet werden:

```
duneM::irCode($code);
```

Das Argument *$code* muss in diesem Fall dem IR-Code entsprechen, der die gewünschte Aktion durchführt, wie auf folgender Webseite dokumentiert:

http://dune-hd.com/support/rc

Beispiel: um die Taste "1" der Fernbedienung zu emulieren, muss folgender Befehl versendet werden:

```
duneM::irCode("F40BBF00");
```

Um die Emulation der Fernbedienung innerhalb der Scripts weiter zu vereinfachen, verfügt die Bibliothek über eine weitere Funktion, welche anstelle des IR-Codes die Bezeichnung der Fernbedienungs-Taste aufnimmt:

```
duneM::irButton($button);
```

In diesem Fall kann die Taste "1" ganz einfach über folgenden Befehl emuliert werden:

```
duneM::irButton("1");
```

Folgende Tabelle gibt eine Übersicht der Funktionen, welche von letzterer Funktion unterstützt werden. Bitte achten Sie auf die korrekte Eingabe der Daten sowie die Groß-/Kleinschreibung. Zudem stellen Sie bitte sicher, dass die Daten immer unter Anführungszeichen übergeben werden (also als STRINGS):

EJECT	MUTE	MODE	POWER	A	B	C	D
1	2	3	4	5	6	7	8
9	0	CLEAR	SELECT	V+	V-	P+	P-
SETUP	UP	DOWN	LEFT	RIGHT	ENTER	RETURN	INFO
PLAY	PAUSE	PREV	NEXT	STOP	REW	FWD	REC

Weitere Informationen zu den verfügbaren Tasten und unterstützten Funktionen finden Sie in der Online-Dokumentation von DUNE.

10.7 WIEDERGABE-STEUERUNG

Die Lautstärke des DUNE-Gerätes kann über folgende Funktion gesteuert werden:

```
duneM::setVolume($volume);
```

Das Argument *$volume* kann eine Ganzzahl zwischen 0 und 100; zudem kann das Gerät über folgenden Befehl stummgeschalten (MUTE, Wert 1) bzw. wieder aktiviert werden (UNMUTE, Wert 0):

```
duneM::setMute($mute);
```

Des Weiteren kann die Wiedergabe des aktuellen Multimedia-Inhalts über folgende Befehle gestartet bzw. pausiert werden:

```
duneM::play();
duneM::pause();
```

11 "MESSAGES" – BIBLIOTHEK

11.1 EINLEITUNG

Die "messages" – Bibliothek ermöglicht die Versendung von OnScreen- und E-Mail-Nachrichten direkt über ein Script, ohne dass dafür zwingend die normalerweise benötigten Objekte innerhalb der VISUALISIERUNG erstellt werden müssen.

11.2 EINBINDUNG DER BIBLIOTHEK

Um die Bibliothek in ein Script einzubinden, reicht folgender Aufruf zu Beginn des Scripts:

```
include_library("messages");
```

Von der nächsten Zeile an kann im Script folgende Klasse verwendet werden:

```
msgM
```

Diese Klasse ("Nachrichten-Manager") stellt diverse Funktionen (bzw. "Methoden") zur Verfügung, um Objekte der Software zu laden und Operationen auszuführen. Die Klasse **msgM** wird, wie bereits bei anderen Klassen gesehen, über "statische Aufrufe" verwendet, wie z.B.:

```
msgM::sendVideoMsg(...);
```

Wie gezeigt, erfolgen die statischen Aufrufe beginnend mit dem Namen der Klasse (ohne das typische Variablen-Präfix "$"), gefolgt von zwei Doppelpunkten und dem Namen der aufzurufenden Funktion; diese Aufrufe unterscheiden sich von normalen Objekt-Aufrufen (deren Name immer das Präfix "$" aufweist), welche das Trennzeichen "->" vor dem Namen der aufzurufenden Funktion benötigen.

11.3 VERSENDUNG EINER ONSCREEN-NACHRICHT

Um eine neue OnScreen-Nachricht innerhalb eines Scripts zu erstellen und zu versenden, kann folgender Befehl verwendet werden:

```
msgM::sendVideoMsg($msg,$level);
```

Wobei

$msg	Text / Inhalt der Nachricht	
$level	Prioritätsstufe der Nachricht; mögliche Werte:	
	0	Alarm (Standard)
	1	Warnung
	2	Information

Beispiel:

```
msgM::sendVideoMsg("Hallo Welt!",2);
```

Natürlich können in der Nachricht auch Werte übergeben werden, z.B. der Wert eines Objekts im EINGANG des verknüpften Run-Scripts:

```
msgM::sendVideoMsg("Eingangswert " . input() . " erhalten!",2);
```

Die Funktion gibt bei erfolgreicher Versendung der Nachricht den Wert _SCRIPT_RESULT_NOERROR zurück, im Falle von Fehlern hingegen die entsprechende Fehlermeldung.

11.4 VERSENDUNG EINER E-MAIL-NACHRICHT

Auf ähnliche Art und Weise wie im letzten Kapitel beschrieben, kann auch eine E-Mail-Nachricht über folgenden Befehl versendet werden:

```
msgM::sendEMailMsg($msg,$to,$subject,$cc,$bcc);
```

Wobei

$msg	Text / Inhalt der Mail	
$to	Mail-Adresse(n) des Empfängers (mehrere Adressen bitte durch ";" trennen)	
$subject	Betreff der Nachricht	[Optional]
$cc	Mail-Adressen in Kopie	[Optional]
$bcc	Mail-Adressen in Kopie (versteckt)	[Optional]

Auch diese Funktion gibt bei erfolgreicher Versendung der Mail den Wert _SCRIPT_RESULT_NOERROR zurück, im Falle von Fehlern hingegen die entsprechende Fehlermeldung.

12 BEISPIELE

12.1 EINLEITUNG

Dieses Kapitel zeigt und erklärt die verschiedenen Beispiel-Scripts, die in Ihrem **KNX**CONTROL-Gerät vorinstalliert sind. Diese können als Basis zur Erschaffung eigener Scripts verwendet werden.

12.2 LOGIC.AND

Dieses Beispiel-Scripts führt eine AND-Verknüpfung auf alle im Eingang verknüpften Objekte des Run-Scripts, welches dieses Script integriert, aus. Der Beispiel-Code ist der folgende (einige Kommentare wurden der Übersicht zuliebe weggelassen):

```
include_library("surrounding");

//Initializing logic result
$result = true;

//Scanning parent relations
foreach($me->parentRelations as $parentRelation)
{
    $parent = $parentRelation->parent;
    $parent_value = $parent->value;

    //Calculating new logic result by putting parent value in AND with previous result
    $result = $result && (!empty($parent_value));
```

```
    //If result is no longer true, skipping: no need for further calculation, it won't
never be true again!
    if(!$result)  break;
}

//Sending result to output - In this way, runner will pass it to child (active event)
output($result);
```

Dieses Beispiel ist jenem aus Kapitel 6.4 dieses Handbuchs sehr ähnlich: auch hier wird zuerst eine Prüfung sämtlicher PARENT-Verknüpfungen (Objekte im EINGANG des Run-Scripts) durchgeführt und von jedem dieser Objekte der aktuelle Wert bestimmt; diese Werte werden im Anschluss verwendet, um sequentiell die AND – Verknüpfung (zum jeweils vorhergehenden Wert) zu bestimmen.

Sobald das Resultat einmal FALSE ist, wird die Schleife abgebrochen, da die AND - Verknüpfung in diesem Moment nicht mehr TRUE werden kann.

Der berechnete Wert wird im Anschluss über die Funktion output() zurückgegeben.

12.3 LOGIC.OR

In derselben Art und Weise wie vorhergehendes Script, führt dieses Beispiel-Script eine OR – Verknüpfung auf alle Objekte im Eingang durch. Die Unterschiede zum vorhergehenden Script sind in rot aufgeführt:

```
include_library("surrounding");

//Initializing logic result
$result = false;

foreach($me->parentRelations as $parentRelation)
{
    $parent = $parentRelation->parent;
    $parent_value = $parent->value;

    //Calculating new logic result by putting parent value in OR with previous result
    $result = $result || (!empty($parent_value));

    //If result is true, skipping: no need for further calculation, it won't never be
false again!
    if($result)  break;
}

//Sending result to output - In this way, runner will pass it to child (active event)
output($result);
```

In diesem Fall wird die Schleife automatisch unterbrochen, wenn eines der Objekte im Eingang ungleich 0 (Null) ist; die OR – Verknüpfung kann in dem Moment nicht mehr FALSE werden und die Weiterführung der Schleife macht wenig Sinn.

12.4 SERIAL.WRITER_GENERIC

Dieses Script zeigt das Beispiel eines Schreibvorganges auf die RS232-Schnittstelle von **KNX**SERVER; als Nachricht wird der INPUT-Wert des Run-Scripts übernommen. Auch in diesem Fall wurden im Beispiel-Code einige der Kommentare der Übersicht zuliebe weggelassen:

```
include_library("serial");

//Uncomment following lines if you need to change default serial settings
//$serialM->deviceSet("/dev/ttyS4");    //ATTENTION: change it at your own risk! This is default RS232 port of KNXSERVER
//$serialM->confBaudRate(19200);     //Change     baudrate     -     Allowed     values: 110,150,300,600,1200,2400,4800,9600,19200,38400,57600,115200
//$serialM->confParity("none");    //Change parity - Allowed values: "none","odd","even"
//$serialM->confCharacterLength(8);//Change character length
//$serialM->confStopBits(1);      //Change stop bits
//$serialM->confFlowControl("none");//Change    flow    control    -    Allowed    values: "none","rts/cts","xon/xoff"

//Opening serial port
$serialM->deviceOpen();

//Initializing message as input value
$msg = input();

//Sending message to serial port, by adding a trailing return at the end (new line)
$serialM->sendMessage($msg . "\n");

//Closing serial port
$serialM->deviceClose();

//Sending input value to output - In this way, value is passed to any eventual runner's child and set as its value
output(input());
```

Wie man erkennen kann, sind vor dem Befehl zum Öffnen der Schnittstelle alle Befehle zur Veränderung der Einstellungen des RS232-Ports aufgeführt. Sollten also Änderungen benötigt werden, reicht es aus, den Kommentar (Präfix "//") der entsprechenden Befehle zu entfernen und den gewünschten Wert einzustellen, wie in Kapitel 7.3 genauer beschrieben.

Der Wert, der an die Schnittstelle gesendet wird, ist der INPUT-Wert (wird über die Funktion input() abgerufen), gefolgt vom Zeichen zur Durchführung eines Zeilensprungs (kann bei Bedarf angepasst werden).

Nach Abschluss des Schreibvorgangs und Schließen der Schnittstelle stellt das Script den gesendeten Wert auch über die Funktion output() bereit, damit das Run-Script – sollte es in der Visualisierung eingebunden sein – diesen Wert auch darstellen kann. Dieser Schritt ist optional und hat mit dem Schreibvorgang selbst nichts zu tun; er kann deshalb wie gewünscht angepasst oder auch entfernt werden.

12.5 DEMUX.STATUSBYTE

Dieses Script führt eine "De-Multiplexing" – Operation auf dem INPUT-Wert des Run-Scripts durch. Es wird vorausgesetzt, dass es sich beim INPUT-Wert um einen 1 Byte – Wert handelt (numerischer Wert zwischen 0 und 255). Dieser wird durch das Script in 8 Werte zu 1 Bit zerlegt, welche im Anschluss im Binär-Format vorliegen.

Dieses Script kann vor allem zur Verarbeitung der sog. "Status-Bytes" nützlich sein, welche von diversen KNX-Geräten verwendet werden, um ihren Status auf den Bus zu senden (daher auch der Name des Scripts). Das Script "entpackt" diesen Wert und erlaubt es, die einzelnen Bit-Werte 8 Objekten im Ausgang des Run-Scripts zuzuweisen, über welche somit der entsprechende Status dargestellt werden kann.

Das Script setzt voraus, dass das ausführende Run-Script 8 Objekte im Ausgang verknüpft hat und weist jedem dieser Objekte – in der Reihenfolge, wie sie im Run-Script eingefügt sind – den entsprechenden Bit-Wert zu, der beim „Zerlegen" des 1 Byte – Wertes entsteht.

```
include_library("surrounding");

//Converting input value to binary
$statusByte = str_pad(decbin(input()),8,'0',STR_PAD_LEFT);

//Scanning childs up to 8th, setting corresponding demuxed value
for($i=0;$i<8;$i++)
{
  //Initializing output name into a temporary variable
  $outName = "out" . ($i+1);

  //If variable is not set, exiting loop - It means we don't have enough children associated to the run-script
  if(!isSet($$outName)) break;

  //Setting value to child
  $$outName->set($statusByte[$i] ? 1 : 0);
}

//Returning an empty string - In such a way, environment won't call children again (since we already did it!)
output();
```

Folgende Zeile führt das eigentliche "De-Multiplexing" durch:

```
$statusByte = str_pad(decbin(input()),8,'0',STR_PAD_LEFT);
```

Der erhaltene numerische Wert wird zuerst in Binär-Code konvertiert (fehlende "Nullen" am Anfang werden automatisch hinzugefügt, um eine fixe Länge von 8 Bit zu erhalten). Im nächsten Schritt werden die Objekte im Ausgang des Run-Scripts (von 1 bis 8) über eine Schleife einer "Platzhalter"-Variablen zugewiesen:

```
$outName = "out" . ($i+1);
```

Dann wird geprüft, ob dieser Ausgang auch wirklich belegt ist; sollte kein Objekt im Ausgang vorhanden sein (z.B. weil im Run-Script nicht verknüpft), wird die Schleife unterbrochen:

```
if(!isSet($$outName)) break;
```

Sind die 8 Objekte im Ausgang hingegen vorhanden, wird ein "Set" – Befehl ausgeführt, über welchen der berechnete Bit-Wert (entweder 0 oder 1) übergeben wird:

```
$$outName->set($statusByte[$i] ? 1 : 0);
```

Die PHP-Sprache erlaubt die Verwendung des Wertes einer Variable wie eine Variable selbst, wie z.B. bei…

$$outName

… der Fall ist. Anstatt den Wert von $outName (= "out1", "out2" usw.) explizit aufzuführen…

$out1, $out2 etc…

> ... kann ebenso $$outName verwendet werden, welcher in jedem Schleifendurchgang dem Wert des referenzierten Objekts im Ausgang des Run-Scripts entspricht (siehe Kapitel 6.3 zur Erklärung von $out).

Zum Schluss wird das Script über folgenden Aufruf beendet:

```
output();
```

Es wird folglich kein Wert zurückgegeben, wodurch verhindert wird, dass die Objekte im Ausgang, welche bereits durch das Script selbst angesteuert wurden, nochmals einen Wert übergeben bekommen.

12.6 MATH.SUM

Dieses Beispiel-Script berechnet die SUMME der Werte aller Objekte im Eingang des Run-Scripts. Es folgt der Beispiel-Code (die Kommentare wurden der Übersicht zuliebe vollständig entfernt):

```
include_library("surrounding");

$sum = 0;

foreach($me->parentRelations as $parentRelation)
{
    $parent = $parentRelation->parent;
    $parent_value = $parent->value;

    $sum = $sum + floatval($parent_value);
}

output($sum);
```

Auch dieses Beispiel gleicht jenem aus Kapitel 6.4 dieses Handbuchs: es wird zuerst eine Prüfung sämtlicher PARENT-Verknüpfungen (Objekte im EINGANG des Run-Scripts) durchgeführt und von jedem dieser Objekte der aktuelle Wert bestimmt; diese Werte werden im Anschluss verwendet, um sequentiell die Summe aller Werte zu bestimmen.
Der so berechnete Wert wird im Anschluss über die Funktion **output()** zurückgegeben.

12.7 MATH.PRODUCT

Dieses Beispiel-Script berechnet das PRODUKT der Werte aller Objekte im Eingang des Run-Scripts. Es folgt der Beispiel-Code (die Kommentare wurden der Übersicht zuliebe vollständig entfernt):

```
include_library("surrounding");

$product = 1;

foreach($me->parentRelations as $parentRelation)
{
    $parent = $parentRelation->parent;
    $parent_value = $parent->value;

    $product = $product * floatval($parent_value);
}

output($product);
```

Das Script ist praktisch identisch mit dem vorhergehenden, mit dem einzigen Unterschied (neben dem "*"-Zeichen selbst anstelle von "+"), dass das Resultat zu Beginn mit dem Wert 1 anstelle von 0 initialisiert werden muss.
Auch in diesem Fall wird der berechnete Wert über die Funktion **output()** zurückgegeben.

12.8 MODBUS.READCOILS

Dieses Beispiel-Script liest eine Serie von Coils (Binär-Ein- bzw. Ausgänge) von einem MODBUS SLAVE – Gerät aus und kann im Anschluss ein im AUSGANG des Run-Scripts verknüpftes Objekt ansteuern:

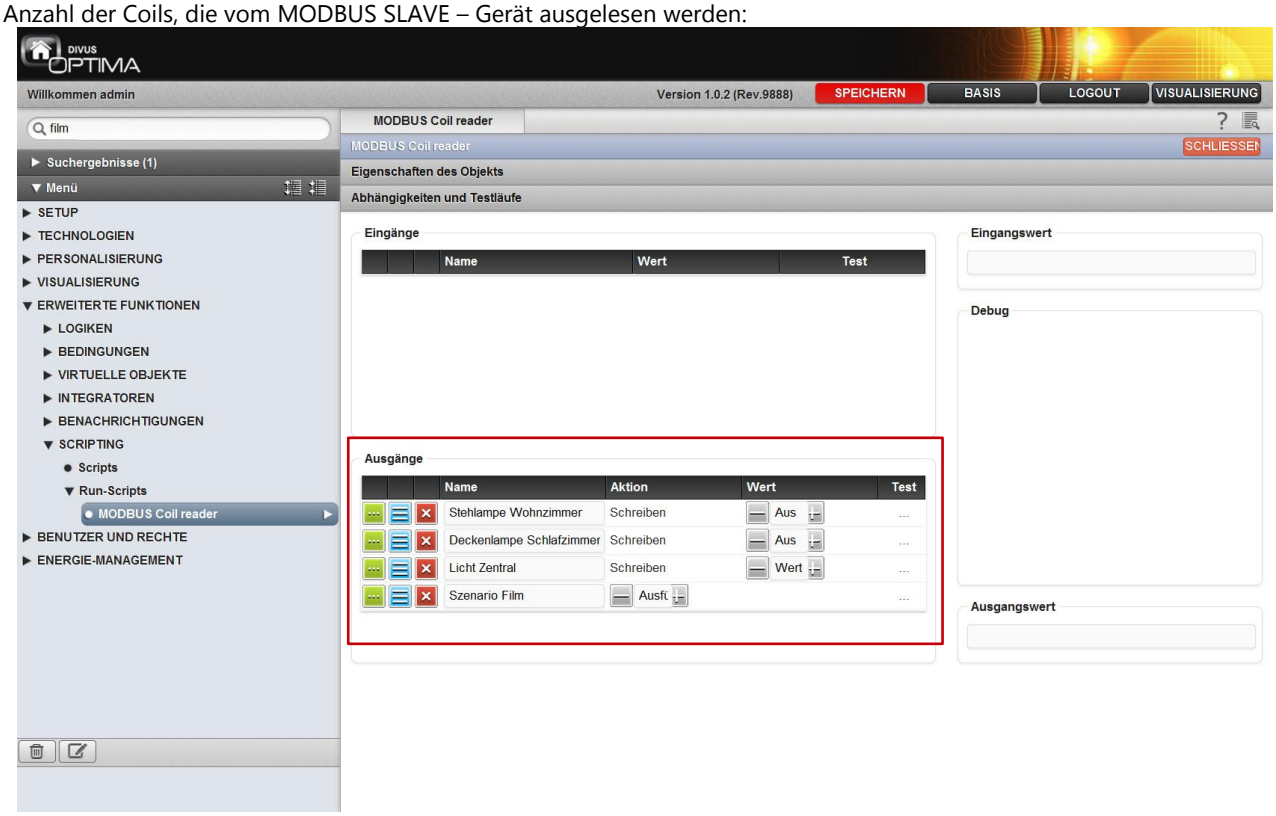
```
include_library("surrounding");

$host = "192.168.0.71";   //Change this according with MODBUS SLAVE IP address
$protocol = "TCP";        //Same as above
$start = 0;               //First coil to read

$number = count($me->childRelations);
if($number>0)
{
 include_library("modbus");

 modbusM::bind($host,$protocol);

 $values = modbusM::readCoils(0, $start, $number);

 for($i=0;$i<count($values);$i++)
 {
  $value = !empty($values[$i]) ? 1 : 0;
  debug("Coil " . ($start + $i) . " has value $value",true);

  $childName = "out" . ($i+1);
  if(is_object($$childName))
  {
   if($$childName->value == $value) continue;

   $$childName->set($value);
   debug("Object " . $$childName->name . " set to $value",true);
  }
 }
}

output();
```

Wie man unschwer erkennen kann, initialisiert der erste Teil des Scripts einige der Eigenschaften, welche zur Verbindung mit dem MODBUS SLAVE – Gerät benötigt werden:

```
$host = "192.168.0.71";   //Change this according with MODBUS SLAVE IP address
$protocol = "TCP";        //Same as above
$start = 0;               //First coil to read
```

Neben der IP-Adresse und dem Protokoll ist es wichtig, die MODBUS – Adresse des ersten auszulesenden Coils anzugeben. Bitte nutzen Sie die Dokumentation Ihres MODBUS – Gerätes, um diese Adresse festzustellen (bitte beachten Sie, dass der Befehl zum Auslesen bereits automatisch ein Offset beinhaltet, und zwar "10000").

Das Script ist dazu gedacht, von einem Run-Script mit so vielen Objekten im AUSGANG gestartet zu werden, wie die Anzahl der Coils, die vom MODBUS SLAVE – Gerät ausgelesen werden:



Das Script liest nach dem Starten die Anzahl der Objekte im AUSGANG des Run-Scripts aus...

```
$number = count($me->childRelations);
```

... und liest (nach Kontrolle, ob zumindest ein Objekt vorhanden ist und ob die "modbus" – Bibliothek eingebunden wurde) genauso viele Coils vom MODBUS SLAVE – Gerät aus:

```
$values = modbusM::readCoils(0, $start, $number);
```

Nun wird das Array mit den Resultaten (welches genauso viele Elemente enthält wie Objekte im AUSGANG verknüpft sind) durchlaufen, der aktuelle Wert in eine Variable geschrieben (entweder "1" oder "0", abhängig vom ausgelesenen Status)...

```
$value = !empty($values[$i]) ? 1 : 0;
```

... und ebenso das entsprechende Objekt im Ausgang einer Variable zugewiesen

```
$childName = "out" . ($i+1);
```

Wie schon im Beispiel-Script in Kapitel 9.5 wird nun über folgenden Ausdruck direkt auf den Ausgang zugegriffen:

```
$$childName
```

Nach Kontrolle, ob das Objekt auch wirklich existiert...

```
if(is_object($$childName))
```

... wird überprüft, ob das Objekt bereits den Status aufweist, der vom MODBUS – Gerät ausgelesen wurde ($value), da in diesem Fall keine Aktion notwendig wäre:

```
if($$childName->value == $value) continue;
```

Sind die Werte unterschiedlich, wird dem Objekt im AUSGANG der vom MODBUS – Gerät ausgelesene Wert übergeben:

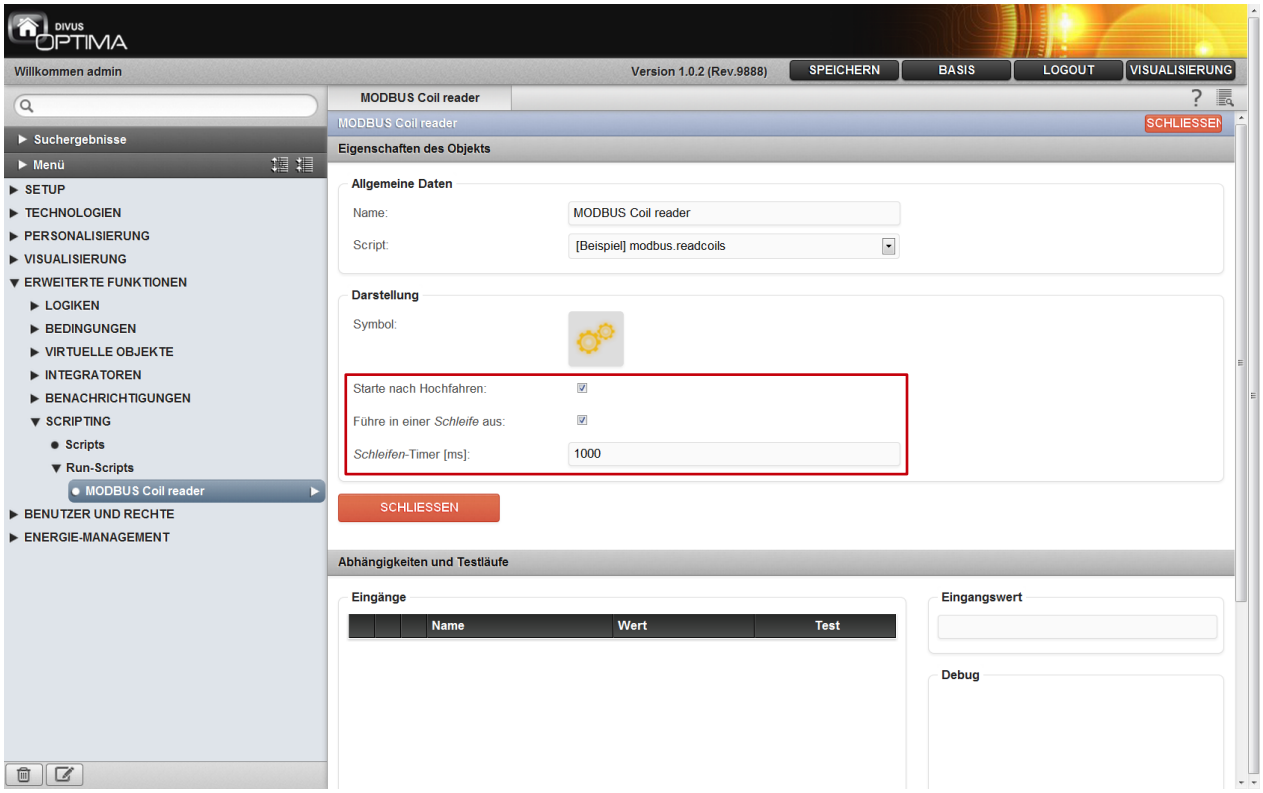
```
$$childName->set($value);
```

Nach Abschluss dieser Aktionen wird das Script ohne Rückgabewert beendet:

```
output();
```

Auch hier ist dieser Schritt sehr wichtig, da die Objekte im AUSGANG ansonsten erneut angesteuert werden und die zuvor gesetzten Werte einfach überschrieben werden würden.

Typischerweise wird ein Run-Script, das mit dieser Art von Script verknüpft wird, sofort nach dem Booten den Systems gestartet und soll im Hintergrund laufen:



Der Schleifen-Timer definiert, in welchen Abständen die Coils des MODBUS – Gerätes ausgelesen werden ("Polling" - Zeit), und bestimmt somit auch die "Reaktionszeit", in welcher eventuelle Statusänderungen des MODBUS – Gerätes von Ihrem **KNX**CONTROL-Gerät erkannt und verarbeitet werden.

Zu geringe Werte für den Schleifen-Timer (kleiner als eine Sekunde) können zu einer Überlastung Ihres **KNX**CONTROL-Gerätes führen, vor allem wenn viele Coils ausgelesen werden sollen (gilt auch für das Auslesen von Registern mit ähnlich aufgebauten Scripts).

12.9 MODBUS.READREGISTERS

Dieses Beispiel-Script ist vom Konzept her identisch aufgebaut wie das vorhergehende, mit dem Unterschied dass es Register anstelle von Coils vom MODBUS SLAVE – Gerät ausliest. Zur leichteren Erkennung wurden die primären Unterschiede zum vorhergehenden Script in Farbe hervorgehoben:

```
include_library("surrounding");

$host = "192.168.0.71";  //Change this according with MODBUS SLAVE IP address
$protocol = "TCP";       //Same as above
$start = 0;              //First register to read
$dataLength = 2;         //Bytes for each data - Typically 2 or 4 bytes

$number = count($me->childRelations);
if($number>0)
{
 include_library("modbus");

 modbusM::bind($host,$protocol);

 $data = modbusM::readRegisters(0, $start, $number);

 $values = array_chunk($data, $dataLength);
 for($i=0;$i<count($values);$i++)
 {
  $value = PhpType::bytes2signedInt($values[$i]);
  debug("Register " . ($start + $i) . " has value $value",true);

  $childName = "out" . ($i+1);
  if(is_object($$childName))
  {
   $$childName->set($value);
   debug("Object " . $$childName->name . " set to $value",true);
  }
 }
}

output();
```

In diesem Fall muss auch angegeben werden, in welchem Format (2 oder 4 Byte) die Daten aus dem MODBUS – Gerät ausgelesen werden und im Anschluss den Objekten im AUSGANG zugewiesen werden sollen:

```
$dataLength = 2;         //Bytes for each data - Typically 2 or 4 bytes
```

Somit kann das erhaltene Byte-Array je nach angegebenem Format zerlegt werden...

```
$values = array_chunk($data, $dataLength);
```

... und im Anschluss Wert für Wert an die Objekte im AUSGANG übergeben werden. Je nach Formatierung der via MODBUS erhaltenen Daten muss ev. eine entsprechende Konvertierung der Daten durchgeführt werden, wie z.B.:

```
$value = PhpType::bytes2signedInt($values[$i]);
```

Weitere Informationen zu den möglichen Konvertierungen finden Sie in Kapitel 8.9 dieses Handbuchs.

Der Einfachheit halber führt dieses Beispiel keine Kontrolle des Wertes der Objekte im AUSGANG vor dem Senden des MODBUS – Wertes durch; es empfiehlt sich auf jeden Fall, diese Kontrolle – wie auch im Beispiel-Script zum Auslesen der Coils – einzufügen (vor allem bei Ausführung in einer Schleife), um zu vermeiden, dass kontinuierlich Werte auf den Bus geschrieben werden, welche u.U. gar nicht notwendig sind.

Auch dieses Beispiel-Script wird im Normalfall über das Run-Script in einer Schleife ausgeführt werden und bereits bei Systemstart ausgeführt werden.

12.10 MODBUS.WRITECOILS

Dieses Beispiel-Script schreibt eine Serie von Coils an ein MODBUS SLAVE – Gerät, in Abhängigkeit von Statusänderungen der im EINGANG des Run-Script verknüpften Objekte:

```
include_library("surrounding");

$host = "192.168.0.71";   //Change this according with MODBUS SLAVE IP address
$protocol = "TCP";        //Same as above
$start = 0;               //First coil to write

$number = count($me->parentRelations);
if($number>0)
{
 include_library("modbus");

 modbusM::bind($host,$protocol);

 $values = array();

 foreach($me->parentRelations as $parentRelation)
 {
  $parent = $parentRelation->parent;
  $value = $parent->value;

  $values[] = (!empty($value) ? TRUE : FALSE);
 }

 $result = modbusM::writeCoils(0, $start, $values);
 debug("Written coils to modbus slave: " . implode(",",$values));
}

output();
```

Wie bereits zuvor gesehen, dient der erste Teil des Scripts der Konfiguration der Kommunikation mit dem MODBUS SLAVE – Gerät:

```
$host = "192.168.0.71";   //Change this according with MODBUS SLAVE IP address
$protocol = "TCP";        //Same as above
$start = 0;               //First coil to write
```

Für dieses Script muss auch der erste zu schreibende Coil (**$start**) angegeben werden. Die Anzahl der zu schreibenden Coils wird durch die im EINGANG des Run-Script verknüpften Objekte bestimmt:

```
$number = count($me->parentRelations);
```

Nach Einbindung der "modbus" – Bibliothek und der Prüfung, ob zumindest ein Objekt als EINGANG definiert wurde, erzeugt das Script ein Werte-Array anhand der Werte der Objekte im EINGANG…

```
foreach($me->parentRelations as $parentRelation)
 {
  $parent = $parentRelation->parent;
  $value = $parent->value;

  $values[] = (!empty($value) ? TRUE : FALSE);
 }
```

… und schreibt dieses an das MODBUS - Gerät:

```
$result = modbusM::writeCoils(0, $start, $values);
```

Auch in diesem Fall gibt das Script keinen Wert zurück und wird nach dem Schreibvorgang beendet.

12.11 SONOS.GENERIC

Folgendes Script beinhaltet ein Demo-Beispiel zur Ansteuerung eines SONOS-Gerätes:

```
include_library("sonos");

sonosM::bind("192.168.0.150");

sonosM::play();
debug("Now SONOS is playing...",true);

sleep(5);

sonosM::setVolume(50);
debug("Now SONOS volume is 50%...",true);

sleep(5);

sonosM::next();
debug("Skipped to next SONOS track...",true);

sleep(5);

sonosM::pause();
debug("Now SONOS is in pause...",true);

output();
```

Nach der Verbindung mit dem SONOS-Gerät mit IP-Adresse 192.168.0.150 sendet dieses Script im 5-Sekunden-Takt verschiedene Befehle an das Gerät, welche ausgeführt werden müssen.

12.12 SONOS.SETVOLUME

Dieses Script setzt die Lautstärke eines SONOS-Gerätes über einen erhaltenen Eingangswert (normalerweise der Wert eines Objektes, welches als EINGANG des Run-Scripts verknüpft worden ist, das dieses Script ausführt):

```
include_library("sonos");
sonosM::bind("192.168.0.150");
$volume = input();
sonosM::setVolume($volume);
output();
```

Dieses Script ist sowohl mit der Funktion "LAUTSTÄRKE" für KNX Objekte als auch mit der Lautstärke-Funktion des komplexen Objektes "MULTIROOM - ZONE" kompatibel.

12.13 SONOS.GETVOLUME

Dieses Script holt sich den aktuellen Lautstärke-Wert von einem SONOS-Gerät und gibt diesen im Ausgang zurück (und gibt ihn somit normalerweise an ein Objekte weiter, welches als AUSGANG des Run-Scripts verknüpft worden ist, das dieses Script ausführt):

```
include_library("sonos");
sonosM::bind("192.168.0.150");
$volume = sonosM::getVolume();
output($volume);
```

12.14 SONOS.PLAYPAUSE

Dieses Script startet die Multimedia-Wiedergabe (PLAY) eines SONOS-Gerätes oder pausiert diese (PAUSE), in Abhängigkeit eines Eingangswertes (1 oder 0):

```
include_library("sonos");
sonosM::bind("192.168.0.150");
$cmd = input();
if(intval($cmd)==1)
 sonosM::play();
else
 sonosM::pause();

output();
```

Dieses Script ist sowohl mit der Funktion "PLAY/PAUSE" für KNX Objekte als auch mit der Play/Pause-Funktion des komplexen Objektes "MULTIROOM - ZONE" kompatibel.

12.15 SONOS.PREVNEXT

Dieses Script springt zum nächsten oder vorhergehenden Track der Wiedergabeliste eines SONOS-Gerätes, in Abhängigkeit eines Eingangswertes (1 oder 0):

```
include_library("sonos");
sonosM::bind("192.168.0.150");
$cmd = input();
if(intval($cmd)==1)
 sonosM::next();
else
 sonosM::previous();
output();
```

Dieses Script ist sowohl mit der Funktion "VOR/ZURÜCK" für KNX Objekte als auch mit der Vor-/Zurück-Funktion des komplexen Objektes "MULTIROOM - ZONE" kompatibel.

12.16 SONOS.SETPLAYMODE

Dieses Script verändert den Wiedergabemodus eines SONOS-Gerätes, in Abhängigkeit eines Eingangswertes:

```
include_library("sonos");
sonosM::bind("192.168.0.150");
$mode = input();
sonosM::setPlayMode($mode);
output();
```

Dieses Script ist sowohl mit der Funktion "WIEDERGABEMODUS" für KNX Objekte als auch mit der Wiedergabemodus-Funktion des komplexen Objektes "MULTIROOM - ZONE" kompatibel.

12.17 SONOS.GETINFO

Dieses Script holt sich vom SONOS-Gerät Informationen zum aktuell abgespielten Track und gibt diese Informationen an verschiedene Objekte weiter, welche als AUSGÄNGE des Run-Scripts definiert worden sind (die Reihenfolge der Wertübergabe wird dabei vom Script definiert, weswegen die Objekte, die den Wert entgegennehmen sollen, in derselben Reihenfolge im Ausgang des Run-Scripts verknüpft werden müssen):

```
include_library("surrounding");
include_library("sonos");
sonosM::bind("192.168.0.150");
$info = sonosM::getMediaInfo();
```

```
if(isSet($out1) && ($out1->value != $info["artist"])) $out1->set($info["artist"]);
if(isSet($out2) && ($out2->value != $info["title"])) $out2->set($info["title"]);
if(isSet($out3) && ($out3->value != $info["album"])) $out3->set($info["album"]);
if(isSet($out4) && ($out4->value != $info["Track"])) $out4->set($info["Track"]);
if(isSet($out5) && ($out5->value != $info["position"])) $out5->set($info["position"]);
if(isSet($out6) && ($out6->value != $info["duration"])) $out6->set($info["duration"]);

output();
```

12.18 DUNE.ONOFF

Dieses Script schaltet ein DUNE-Gerät entweder aus oder ein, in Abhängigkeit des erhaltenen Eingangswertes (entweder 0 oder 1):

```
include_library("dune");
duneM::bind("192.168.0.150");
$cmd = input();
if(intval($cmd)==1)
 duneM::mainScreen();
else
 duneM::standby();

output();
```

12.19 DUNE.PLAYPAUSE

Dieses Script startet die Multimedia-Wiedergabe (PLAY) eines DUNE-Gerätes oder pausiert diese (PAUSE), in Abhängigkeit eines Eingangswertes (1 oder 0):

```
include_library("dune");
duneM::bind("192.168.0.150");
$cmd = input();

if(intval($cmd)==1)
 duneM::play();
else
 duneM::pause();

output();
```

Dieses Script ist sowohl mit der Funktion “PLAY/PAUSE” für KNX Objekte als auch mit der Play/Pause-Funktion des komplexen Objektes “MULTIROOM - ZONE” kompatibel.

12.20 DUNE.SETVOLUME

Dieses Script setzt die Lautstärke eines DUNE-Gerätes über einen erhaltenen Eingangswert:

```
include_library("dune");
duneM::bind("192.168.0.150");

$volume = input();
duneM::setVolume($volume);

output();
```

Dieses Script ist sowohl mit der Funktion "LAUTSTÄRKE" für KNX Objekte als auch mit der Lautstärke-Funktion des komplexen Objektes "MULTIROOM - ZONE" kompatibel.

12.21 DUNE.IRCOMMAND

Dieses Script emuliert einen Tastendruck auf der Fernbedienung des DUNE-Systems, in Abhängigkeit des erhaltenen Eingangswertes:

```
include_library("dune");
duneM::bind("192.168.0.150");

$button = input();
duneM::irButton($button);

output();
```

12.22 DEWPOINT

Dieses Script berechnet den Taupunkt in Abhängigkeit einer TEMPERATUR und eines relativen FEUCHTIGKEITSWERTS, welche als INPUTS des Run-Scripts verknüpft werden müssen. In untenstehendem Beispiel wird davon ausgegangen, dass die Temperatur als 1. Wert und die Feuchtigkeit als 2. Wert verknüpft worden ist:

```
include_library("surrounding");

$T = floatval($in1->value);
$H = floatval($in2->value);

$D = round(pow(($H/100),(1/8)) * (112 + (0.9 * $T)) + (0.1 * $T) - 112,1);
output($D);
```

Die Berechnung basiert auf der Näherungsformel, welche auf folgender Seite beschrieben wird:
http://de.wikipedia.org/wiki/Taupunkt

13 ANHANG

13.1 FUNKTIONS-OBJEKTTYPEN

Die Tabelle auf der folgenden Seite zeigt eine Auflistung sämtlicher Funktionstypen der von der OPTIMA-Oberfläche vorgesehenen Objekte. Diese werden vom Kommunikationsdienst verwendet, um sämtliche Operationen auf LowLevel-Ebene der VISUALISIERUNG durchzuführen. Normalerweise werden diese Objekttypen in der Software mit dem Kürzel "TYPE" referenziert.

Die Tabelle enthält für jeden Objekttyp neben einer Beschreibung auch die ausführbaren Aktionen; diese können über folgenden Befehl aufgerufen werden:

```
objM::objPerformOperation(ID,OPERATION,VALUE);
```

Details zur korrekten Verwendung dieser Funktion finden Sie in diesem Handbuch in Kapitel 5.5.

TYP ("TYPE")	BESCHREIBUNG	AKTIONEN
VIRTUALOBJECT	Virtuelles Objekt	SETVALUE → Setzt den Wert des Objekts
SCENARIO	Szenario	EXECUTE → Führt das Szenario aus STOPEXECUTION → Unterbricht das Szenario
LOGIC	Logik	EVALUATE → Evaluiert die Logik
CONDITION	Bedingung	CALCULATE → Berechnet die Bedingung
CLIENTBROWSER	Client	REDIRECT → Führt einen Seiten-Sprung auf dem Client-Gerät aus VOIPCALL → Ruft das entsprechende Client-Gerät an
NOTIFY_VIDEO	OnScreen-Nachricht	INSERT → Fügt eine OnScreen-Nachricht ein
NOTIFY_EMAIL	Mail-Nachricht	SENDMAIL → Versendet eine E-Mail-Nachricht
INTEGRATOR	Integrator	INTEGRATE → Aktualisiert die Berechnung des Integrators RESET → Setzt den Integrator auf 0 zurück
EIBOBJECT	KNX Objekt	SETVALUE → Schickt einen Wert an die Gruppenadresse des KNX Objekts GETVALUE → Schickt eine Status-Anfrage an die Gruppenadresse des KNX Objekts
CSCMD	Run-Script	RUN → Führt das Script aus QUIT → Unterbricht das Script
USER	Benutzer	REDIRECT → Führt einen Seiten-Sprung auf allen Geräten aus, auf denen der Benutzer aktiv ist VOIPCALL → Ruft jedes Client-Gerät an, auf welchen der Benutzer aktiv ist
USERGROUP	Benutzergruppe	
USERPERMISSION	Berechtigung	
CONTAINER	Komplexes Objekt	
CAMERA	IP-Kamera	
PBXELEMENT	VoIP-Teilnehmer Außenstelle Rufgruppe	VOIPCALL → Ruft das entsprechende VoIP-Objekt an

13.2 WEB-OBJEKTTYPEN

Folgende Tabelle zeigt eine Auflistung sämtlicher Webtypen der in der OPTIMA-Oberfläche vorgesehenen Objekte. Diese Objekttypen werden von der Web-Oberfläche zur grafischen Darstellung der zuvor gesehenen Funktionstypen verwendet und werden in der Software mit dem Kürzel "PHPCLASS" referenziert.

Für jeden Webtyp wird – insofern vorhanden – der entsprechende Funktionstyp (oder "TYPE", siehe vorhergehendes Kapitel) angegeben; es ist auch möglich, dass mehr als eine PHPCLASS mit demselben „TYPE" verknüpft ist. Ist das Feld „TYPE" leer, so hat die entsprechende PHPCLASS keinerlei Verknüpfung zum Kommunikationsdienst der Software und ihr Einsatz dient einzig der VISUALISIERUNG (z.B. RÄUMLICHKEITEN, LINKS, …)

WEBTYP ("PHPCLASS")	**BESCHREIBUNG**	**FUNKTIONSTYP ("TYPE")**
dpadObject	Allgemeines Objekt*	
dpadGroup	Räumlichkeit (bzw. Ordner / Container / Gruppe)	
dpadUrl	Link	
dpadSysCmd	Systembefehl*	SYSCMD
dpadSysCmdWait	Wartebefehl eines Szenarios	SYSCMD
dpadUser	Benutzer	USER
dpadUserGroup	Benutzergruppe	USERGROUP
dpadUserPermission	Berechtigung einer Benutzergruppe	USERPERMISSION
dpadTriggerObject	Trigger für Datenbank-Änderungen*	TRIGGEROBJECT
dpadContainer	Komplexes Objekt	CONTAINER
dpadVirtualObject	Virtuelles Objekt	VIRTUALOBJECT
dpadScenario	Szenario	SCENARIO
dpadPbxExtension	VoIP Teilnehmer	PBXELEMENT
dpadPbxTrunk	Telefonlinie	
dpadPbxQueue	Rufgruppe	
dpadPbxDoorOpener	Außenstelle	
dpadLogic	Logik	LOGIC
dpadCondition	Bedingung	CONDITION
dpadClientBrowser	Client	CLIENTBROWSER
dpadNotifyVideo	OnScreen-Nachricht	NOTIFY_VIDEO
dpadNotifyEmail	Mail-Nachricht	NOTIFY_EMAIL
dpadIntegrator	Integrator	INTEGRATOR
dpadEibObject	KNX Objekt	EIBOBJECT
dpadCamera	IP-Kamera	CAMERA
dpadScriptRunner	Run-Script	CSCMD
dpadLoadControl	Lastkontrolle	CSCMD

** Objekte, die in der Visualisierung nicht dargestellt werden und nur im Hintergrund arbeiten*

I. RELEASE NOTES

VERSION 1.1:

NEUIGKEITEN SCRIPTING-MODUL:

- Folgende neue Bibliotheken wurden hinzugefügt:
  - SONOS (Audio-Multiroom-Steuerung)
  - DUNE (Audio/Video-Multiroom-Steuerung)
  - MESSAGES (Versendung Nachrichten)
- Erweiterung der OBJECT-Bibliothek (weitere Attribute zugänglich)
- Einführung neuer Beispiel-Scripts

II. NOTIZEN